7.4 Special Purpose Functions

7.4.1 $zb(arg) remove blanks
7.4.2 $zchdir(directory_path) change directory
7.4.3 $zCurrentFile Current Mumps File
7.4.4 $zdump(filename) dump global arrays
7.4.5 $zrestore([arg]) restore globals
7.4.6 $zfile(arg) file exists test
7.4.7 $zflush flush Btree buffers
7.4.8 $zgetenv(arg) get environment variable
7.4.9 $zhtml(arg) encode HTML string
7.4.10 $zhit global array cache hit ratio
7.4.11 $zlower(string) convert to lower case
7.4.12 $znormal(arg1[,arg2]) word normalization
7.4.13 $zNoBlanks(arg) remove all blanks
7.4.14 $zpadd(arg1,arg2) left justify with padding
7.4.15 $zseek(arg)
7.4.16 $zsrand(arg)
7.4.17 $zstem(arg)
7.4.18 $zsystem(arg)
7.4.19 $ztell
7.4.20 $zu(expression)
7.4.21 $zwi(arg)
7.4.22 $zwn extract words from buffer
7.4.23 $zwp extract words from buffer
7.4.24 $zws(string) initialize internal buffer
7.4.25 Scan Functions
7.4.25.1.1 $zzScan
7.4.25.1.2 $zzScanAlnum
7.4.25.1.3 $zzInput(var)

7.5 Vector and Matrix Functions

7.5.1 $zzAvg(vector)
7.5.2 $zzCentroid(gblMatrix,gblRef)
7.5.3 $zzCount(gblVector)
7.5.4 $zzMax(gbl)
7.5.5 $zzMin(gbl)
7.5.6 $zzMultiply(gbl1,gbl2,gbl3)
7.5.7 $zzSum(gblVector)
7.5.8 $zzTranspose(gblMatrix1,gblMatrix2)

7.6 Text Processing Functions

7.6.1 Similarity Functions
7.6.1.1 $zzCosine(gbl1,gbl2)
7.6.1.2 $zzSim1(gbl1,gbl2)
7.6.1.3 $zzDice(gbl1,gbl2)
7.6.1.4 $zzJaccard(gbl1,gbl2)
7.6.2 $zzBMGSearch(arg1,arg2)
7.6.3 $zPerlMatch(string,pattern)
7.6.4 $zReplace(string,pattern,replacement)
7.6.5 $zShred(string,length)
7.6.6 $zShredQuery(string,length)
7.6.7 $zzSoundex(s1)
7.6.8 $zSmithWaterman(s1,s2,align,mat,gap,noMatch,match)
7.6.9 $zzlDF(global,doccoun)
7.6.10 Correlation Functions
7.6.10.1 $zzTermCorrelate(global1,global2)
7.6.10.2 $zzDocCorrelate(gblref1,gblref2,mtdh,thrshld)

7.6.11 Stop and Synonym Functions
7.6.11.1 $zStopInit(arg)
7.6.11.2 $zStopLookup(word)
7.6.11.3 $zSynInit(fileName)
7.6.11.4 $zSynLookup(word)
7.7 SQL FUNCTIONS ............................................................................................................................ 66
  7.7.1 $zsql ........................................................................................................................................... 66
  7.7.2 $zsqlCols .................................................................................................................................. 66
  7.7.3 $zsqlOpen ................................................................................................................................ 67

8 Pattern Matching ............................................................................................................................. 68
  8.1 MUMPS 95 PATTERN MATCHING ............................................................................................. 68
  8.2 USING PERL REGULAR EXPRESSIONS ...................................................................................... 68

9 Mumps interpreter for Windows ...................................................................................................... 70

Index of Figures

Figure 1 Subroutine/Function Calls ..................................................................................................... 46
Figure 2 - $Zb() examples .................................................................................................................. 52
Figure 3 - $Zseek() examples ............................................................................................................. 54
Figure 4 - $Zwi() examples .................................................................................................................. 55
Figure 5 - Scan functions examples .................................................................................................... 56
Figure 6 - $zzAvg() example ............................................................................................................... 56
Figure 7 - $zzCentroid() example ...................................................................................................... 57
Figure 8 - $zzCount() example .......................................................................................................... 57
Figure 9 - $zzMax() example ............................................................................................................... 57
Figure 10 - $zzMin() example ............................................................................................................ 58
Figure 11 - Similarity formulae ........................................................................................................... 59
Figure 12 - Similarity functions .......................................................................................................... 59
Figure 13 - $zzBMGSearch() example ............................................................................................... 60
Figure 14 - $zzMultiply() example ................................................................................................... 60
Figure 15 - $zzSum() example ............................................................................................................ 61
Figure 16 - $zzTranspose() example .................................................................................................. 61
Figure 17 - $zPerlMatch() example ................................................................................................... 61
Figure 18 - $zReplace() example ...................................................................................................... 62
Figure 19 - $zShred() example ........................................................................................................... 62
Figure 20 - $ShredQuery() example .................................................................................................. 63
Figure 21 - $zSmithWaterman() ........................................................................................................ 64
Figure 22 - $zzIDF() example ............................................................................................................. 64
Figure 23 - $zTermCorrelate() ........................................................................................................... 65
Figure 24 - $zDocCorrelate() ............................................................................................................ 66
Figure 25 - Stop list functions ............................................................................................................. 67
1 Installation

1.1 WARNING

Please do not use the Mumps compiler. It has not been updated and there are possible errors. If you insist upon using it, do not send error reports. It will be brought up to date in a later release.

Use the Mumps interpreter instead.

1.2 Required Software

Building mumps requires:

1. Linux, preferably a Debian based version such as Debian, Ubuntu or Mint.
2. The g++/gcc compilers and related libraries.
3. The pcrc (Perl Compatible Regular Expression) development libraries. The pcrc libs should be in /usr/lib and the include files in /usr/include. Be certain to install the pcrc development libraries.
4. PostgreSQL (optional) to store global arrays including the client development libraries.
5. The bash shell interpreter located in /bin.
6. autoconf.

1.3 Configure Options

The basic install sequence, as is the case with many Linux based packages is:

    configure
    make
    make install

The configure step, however, as is typical, contains many options. Specifying these causes modification to the source code. The options to configure are:

1. configure prefix=/usr

   The directory where the runtime modules will be stored. If this is not specified, the default location is in a directory named mumps_compiler in the user's home directory. Normally, if you want Mumps available to all users, you will specify the option as shown and run make and make install as root. If you specify /usr as shown, the Mumps routines will be placed in /usr/bin/mumps.

2. PostgreSQL options

   a) --with-pgsql-user=userid     userid for PostgreSQL server account [mumps]
   b) --with-pgsql-password=password password to access database [“”]
   c) --with-pgsql-host=IPnbr      select host IP number
   d) --with-pgdb=/usr/include/postgresql location of libraries

3. MySQL options

   a) --with-mysql-user=userid     userid for MySQL server account [okane]
   b) --with-mysql-host=nbr        IP number of remote host [localhost]
   c) --with-mysql-db              Enable MySQL data base for globals
   d) --with-mysql-passwd=val      Select mysql user passwd [””]
   e) --with-mysql-port=nbr        Select mysql port [””]
   f) --with-mysql-socket=nbr      Select mysql socket [””]

4. General Relational Database Options

   a) --with-dname=name            SQL data base name [mumps]
   b) --with-indexsize=number      SQL DB index max [64]
   c) --with-tabsize=nbr           number of columns in SQL table [10]
5. Native Database Options

a) --with-cache=VAL native globals cache size [65537]

The only legal values for this parameter are:

9
17
33
65
129
257
513
1025
2049
4097
8193
16385
32769
65537
131073
262145
524289
1048577

b) --with-block=VAL native btree block size [8192]

The native Btree database consists of two files: the tree file (*key.dat*) containing the actual Btree and the data file (*data.dat*) containing stored data. The maximum size of the Btree file is dependent on the block size. The block sizes listed below each have a PAGE SHIFT value and this ultimately determines the maximum file size as shown. The basic internal disk address is effectively 31 bits (signed 32 bit quantity) but, depending upon the block size, some number of bits at the low-order end are always zero. For example, if the block size is 1024, the final 10 bits of an address are always zeros. As only the significant 31 bits are stored, the true address is not 31 bits but 41 bits thus a file size of 2 terabytes is possible.

The only legal values for this parameter are:

1024
2048
4096
8192
16384
32768
65536
131072
262144

The block size determines the internal PAGE SHIFT factor:

<table>
<thead>
<tr>
<th>Block Size</th>
<th>PAGE SHIFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>10</td>
</tr>
<tr>
<td>2048</td>
<td>11</td>
</tr>
<tr>
<td>4096</td>
<td>12</td>
</tr>
<tr>
<td>8192</td>
<td>13</td>
</tr>
<tr>
<td>16384</td>
<td>14</td>
</tr>
<tr>
<td>32768</td>
<td>15</td>
</tr>
<tr>
<td>65536</td>
<td>16</td>
</tr>
<tr>
<td>131072</td>
<td>17</td>
</tr>
<tr>
<td>262144</td>
<td>18</td>
</tr>
<tr>
<td>524288</td>
<td>19</td>
</tr>
<tr>
<td>1048576</td>
<td>20</td>
</tr>
</tbody>
</table>
PAGE SHIFT 10 corresponds to MBLOCK 1024 and a max Btree file size of 2 TB
PAGE SHIFT 11 corresponds to MBLOCK 2048 and a max Btree file size of 4 TB
PAGE SHIFT 12 corresponds to MBLOCK 4096 and a max Btree file size of 8 TB
PAGE SHIFT 13 corresponds to MBLOCK 8192 and a max Btree file size of 16 TB
PAGE SHIFT 14 corresponds to MBLOCK 16384 and a max Btree file size of 32 TB
PAGE SHIFT 15 corresponds to MBLOCK 32768 and a max Btree file size of 64 TB
PAGE SHIFT 16 corresponds to MBLOCK 65536 and a max Btree file size of 128 TB

The data file may grow to a max of $2^{64}$ bytes for all settings.

c) --with-client build native btree client data base code
d) --with-server-dir native Btree server home directory [/etc/mumps]
e) --with-readonly native database will be readonly

6. --with-ibuf=VAL max size interpreted program [32000]
7. --with-strmax=VAL max internal string size [4096]
8. --with-msvc builds code for MSVC MS Windows environment
9. --with-solaris host system is Solaris
10. --with-locale=locale locale information [en_US.UTF-8]
11. --with-terminate-on-error halt interpreter on error [off]
12. --with-includes=DIR to identify header dirs (Apple)
13. --with-libraries=DIR to identify libs (Apple)

1.4 Build Scripts

There are several scripts in the distribution that will build various versions of the system. You must be root to run these.

1. buildClientServer.script

Builds a client-server version using the native Btree. Two binary executable programs are built: mumpsd which is the database server and mumps which is the interpreter client. See below for details.

2. BuildMumpsWithGlobalsInNative.script

Builds a stand-alone mumps interpreter that uses the single-user native Btree. This version has the fastest database, however, a given set of globals (key.dat and data.dat) may only be used by one instance of the interpreter at a time.

3. BuildMumpsWithGlobalsInMySQL.script

Builds a client mumps interpreter that stores its global arrays in a MySQL server. Requires a running MySQL server and user account authorization.

4. BuildMumpsWithGlobalsInPostgreSQL.script

Builds a client mumps interpreter that stores its global arrays in a PostgreSQL server. Requires a running PostgreSQL server and user account authorization.

5. BuildNativeClientServer.script

Builds the native client server configuration in /etc/mumps/. May not be used with Windows. Globals are stored in a common key.dat and data.dat

6. BuildMumpsWithGlobalsInMySQLCygwin.script
BuildMumpsWithGlobalsInNativeCygwin.script

Builds mumps.exe in a Windows based Cygwin environment that stores the globals in either the native Btree or a MySQL database. The mumps.exe file may be used in a normal Command
Prompt window if the appropriate Cygwin DLLs are accessible. These are:

cygcrypto-1.0.0.dll
cygpcre-1.dll
cygstdc++-6.dll
cygz.dll
cygmysqlclient-18.dll
cygssl-1.0.0.dll
cygwin1.dll
cygcc_s-1.dll

The *cygmysqlclient-18.dll* is only needed for the MySQL build.

The Linux builds above may also be invoked with:

```
Build.script option
```

where *option* is one of the following keywords:

- native
- mysql
- postgresql
- clientserver

Note: the *clientserver* option is not available in Windows.

The Windows version should be used, at present, to run stored programs such as:

```
mumps prog.mps
```

1.5 Overview of Database Build Options

Installation involves compiling and linking the source code modules. The main options concern storage of the global arrays:

1. Store the global arrays in the native B-tree data base
   a) single user only version
   b) multi-user version

2. Store the global arrays in a PostgreSQL data base
   a) in a local PostgreSQL server
   b) in a remote PostgreSQL server.

3. Store the global arrays in a MySQL data base
   a) in a local MySQL server
   b) in a remote MySQL server.

The distribution contains code for a native B-tree implementation of the global array facility. This version quite fast with a minimum of overhead and it can efficiently manage very large databases.

It is, however, sensitive to system errors as it does a minimum of checkpointing and maintains a large part of the global array tree in volatile memory. If the system crashes or the program using the global arrays terminates unexpectedly, the contents of the global array are likely lost. However, in applications where speed is important and, in the event of a crash, the program can be re-run without loss of data, they are a good choice.

Option 1.a runs a single user global array facility. The global arrays are stored in one directory, usually the one in which the Mumps program is itself running. Only one program may access the global arrays in a given directory at a time however other Mumps programs may be run concurrently in other directories operating on other global array data sets. Option 1.a is the fastest but most restrictive option.
Option 1.b involves running a central global array server. The server takes global array access
requests by means of internal operating system pipes from one or more Mumps client programs
operating concurrently. This option is slightly slower due to additional system overhead but it
permits multiple concurrent program execution. Only clients running on the same machine as the
server may access the server. The server's global array files are slightly better protected as the
server is not affected by client program crashes and the server does periodic flushing of it's buffers
and journalling is possible. Synchronization of array access is more tricky and requires use of the
awkward Mumps lock command.

Options 2.a, 2.b, 3a and 3b store the global arrays in a relational database. In options 2.a and 3.a
the server is local to the machine running the client Mumps programs and in options 2.b and 3.b,
the server is running on a machine remote to some or all of the Mumps clients. Options 2.b and 3.b
permits Mumps clients to run on the same machine as the server.

Both options 2 and 3 are much slower than option 1.a, the fastest option. However, using a
relational database has significant advantages with regard to reliability and flexibility:

1. All database transactions are ACID (Atomicity, Consistency, Isolation, Durability) compliant.
2. SQL commands such as Begin Transaction, Commit and Rollback are available.
3. The Mumps global arrays can be queried by other programs with SQL commands.
4. Views of the Mumps database are possible.
5. The Mumps global array database can be remote and distributed.

An additional option exists when using the native global array facility. This option enables a
Mumps program to access a PostgreSQL server and conduct SQL commands on that server. This is
discussed below.

1.6 Native Database Resident Global Array Builds

1.6.1 Installation: Single User Native Global Arrays

As root type:

./configure prefix=/usr
make
sudo make install

This builds the stand-alone, single user version of the mumps interpreter and mumps compiler
libraries.

After prefix= you give the name of the directory you want the files installed under. If none, it will
build a directory named mumps_compiler in your home directory. However, if you put the files in
your home directory, you will need to reference ~/mumps_compiler/bin/mumps to run programs. If
you put the files in /usr, they will be available in the default search PATH. Using /usr is preferred
unless you do not have root privileges.

1.6.2 Installation: Native Global Arrays in Client/Server Mode

Beginning in version 13, a Mumps native global array file server has been available. In previous
versions, Mumps programs required exclusive access to the global array database with which they
were working. Consequently, only one Mumps program is permitted to access the database
concurrently. Version 13 included a facility to run the native database as a server supporting
multiple concurrent Mumps clients.

The Mumps global array server deamon permits multiple Mumps client programs to operate
concurrently. There is approximately a 30% performance penalty on global array accesses through
the daemon as opposed to using the stand-alone version.

To build the client-server configuration, first execute, as root, the script:

sudo ./buildClientServer.script
This will create the global array libraries so that global array accesses will be made through the Mumps server (`mumpsd`).

By default, the Mumps server will be placed in `/etc/mumps`.

Next, change directory to `/etc/mumps` and start the daemon, as root, with the command

```
sudo ./mumpsd > log &
```

To halt the server, send it a `SIGINT (^C)` such as:

```
kill -2 1234
```

where 1234 is the `mumpsd` process id. If you do not start the Mumps server daemon, global array access will be unavailable.

Note: during normal Linux shutdown, this signal is sent to each process so `mumpsd` will be properly shutdown in the event of Linux system shutdown. Thus, if you do a proper system shutdown, the server daemon will be correctly terminated. However, failure to properly close the daemon will result in data loss.

The server communicates with the clients by means of sockets. The Mumps global array data base will be in `/etc/mumps`.

A slight performance increase may be gained by attaching the demon to one CPU in a multi-core environment. To do so, use the following command:

```
sudo schedtool -a 0x1 PID
```

where PID is the process id of the demon and 0x1 means the first (cpu0) processor core.

### 1.6.3 Read Only Mode

Normally, in non-server native mode, only one process may access the global array database files at a time\(^1\). Any other processes wait until the current process terminates. While this mode results in very fast global array access, it is not really necessary to block concurrent database access if the database is only read and never written.

Consequently, in the `BuildMumpsWithGlobalsInNative.script`, two versions of the interpreter are built: `mumps` and `mumpsRO`. In the `mumpsRO` version, the database may only be read, not written. Any attempt to write will fail. However, multiple processes using the `mumpsRO` interpreter may concurrently access the database. The `mumpsRO` interpreter may be accessed with the first line code:

```
/usr/bin/mumpsRO
```

A read-only instance may not be used concurrently with any read-write instance. It may only be used concurrently with other read-only instances.

### 1.7 Relational Database Server Resident Global Arrays

Global arrays may be stored in relational database systems. The two currently supported are MySQL (Oracle Corporation) and PostgreSQL (PostgreSQL Global Development Group). With simple code changes, other servers could also be accommodated.

There are advantages and disadvantages to storing globals arrays in a relational database. The hierarchical nature of the Mumps database is ordinarily not well suited to the tabular structure of a relational database and access is slower. On the other hand, relational databases provide flexible multi-user, robust, fully ACID (Atomicity, Consistency, Isolation, Durability) compliant data storage along with a complete suite of transaction processing functions not otherwise available in the Mumps language definition. A further advantage is that global array data may be interrogated and manipulated by ordinary, standard SQL commands.

---

\(^1\) In native server mode, multiple clients may access the server concurrently.
By default, the Mumps processor maps global array references to a 12 column relational database table named `mumps` (this my be changed as may the number of columns). The first column is the name of the global array and has the column name of `gbl`. Columns two through eleven are named `a1, a2, ... a10` and contain the indices from a global array reference. Column twelve is named `a11` and contains the value, if any, stored at the global array reference. For example, the code:

```
set `^birds(1,2,3,4,5)=“ducks”
```

would map to a tuple in the `mumps` database as:

```
+---------------------------+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+
<table>
<thead>
<tr>
<th>gbl</th>
<th>a1</th>
<th>a2</th>
<th>a3</th>
<th>a4</th>
<th>a5</th>
<th>a6</th>
<th>a7</th>
<th>a8</th>
<th>a9</th>
<th>a10</th>
<th>a11</th>
</tr>
</thead>
<tbody>
<tr>
<td>birds</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>(null)</td>
<td>(null)</td>
<td>(null)</td>
<td>(null)</td>
<td>(null)</td>
<td>ducks</td>
</tr>
</tbody>
</table>
+---------------------------+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+
```

By default, the columns `gbl` and `a1` through `a10` are declared `varchar(255)` and `a11` is `varchar` set to the Mumps system maximum string length (default: 4096). The size of columns `gbl` and `a1` through `a10` can be set to other values by `configure`. Smaller values should improve performance. The total number of columns is can also be set by a `configure` option. Data stored in the `mumps` table can be queried by ordinary SQL such as:

```
select all from mumps where gbl='birds' and a1='1' and a2='2';
```

**1.7.1 Database User**

Relational database servers will only accept requests from authorized users. A Mumps program will require that the userid used to access the database be authorized to create, delete, read and write databases. This access control is handled differently depending on the database system. The default user in `configure` is `mumps`.

**1.7.2 Common Relational Database Configure options**

The following `configure` options are common to all relational database backends:

**1.7.2.1 --with-datasize=numeric-value**

The maximum length of a string stored at a global array node. Performance is improved if this value is as small as possible. If an element stored at a global array node exceeds this length, it will be truncated. Default: 512.

**1.7.2.2 --with-dbname=name**

The name of the database (not the database table) in the relational database server. Default: `mumps`

**1.7.2.3 --with-indexsize=numeric-value**

Specify the size of the `varchar` declaration of columns `gbl` and `a1` through `a10`. This is the maximum string length of any individual global array index element. For example, if your global array reference is: `^a(1,2,3)`, you have four index elements: the name of the global array and three indices. Each may have a maximum length, by default, of 64 characters. Performance is improved if this value is as small as possible. If an individual element of a global array index exceeds this length, it will be truncated. Default: 64.

**1.7.2.4 --with-table=name**

Specify the name of the Mumps database table (not the database) in the relational database. This is the table in which the global arrays will be stored. You may have multiple tables in the database such as `labs, medications`, and so forth. Each table may have multiple global arrays. The global arrays in one table are distinct from the global arrays in other tables even though they have have the same global array name. These tables may be queried by SQL statements. The name of the table may be changed dynamically during execution by setting the `$ztable` system variable. The current table in use may be determined by querying `$ztable`. Default: `mumps`.
1.7.2.5 --with-tabsize=number

Maximum number of index elements in a global array reference not counting the name of the
global array itself. Default: 10. This is the maximum depth of any global array tree. If your
application will never exceed a lower value, use that value. The smaller this value, the fewer
columns in the mumps database table and the better the overall performance. The maximum value
is 31.

1.7.3 Postgresql

1.7.3.1 Installation: PostgreSQL Options

Mumps permits storage of global arrays in PostgreSQL database tables. Using this option gives
the Mumps user fully ACID (Atomicity, Consistency, Isolation, Durability) compliant database but
database access will be slower overall.

This version of Mumps supports PostgreSQL access with the global arrays are stored in the
PostgreSQL database.

When you create/store global arrays, they will be stored, by default, in a table known as mumps
on the PostgreSQL server. This table may also be accessed from non-Mumps clients by SQL SELECT
and related statements. When using the PostgreSQL server, it is also possible to construct views of
RDBMS ordinary database tables that can be directly accessed by Mumps as though they were
global arrays. An example of this is given below.

When storing global arrays on a PostgreSQL server it may be desirable, when beginning a lengthy
series of create/store/updates to the global array database, to initiate an SQL transaction with the
SQL BEGIN TRANSACTION; command and ultimately followed by an SQL COMMIT; command. This
will permit the Mumps global array create/store/updates to run faster. However, should there be a
failure before the final COMMIT, the uncommitted data may be lost.

1.7.3.2 PostgreSQL Installation

Note: as of this writing, the PostgreSQL release is 9.3 and this number is used in the following
documentation. Check which version you have and adjust the following accordingly. Subsequent
version of PostgreSQL may have different interfaces and may use different libraries which may
invalidate some or all of the following.

The following setup instructions apply to Ubuntu and Ubuntu-like distros such as Mint. In Red Hat
based distros, some files PostgreSQL files may be located in different directories and this may be
affect the installation procedures.

1. Using your package manager (e.g., Synaptic) install the latest version of PostgreSQL including
the development libraries. This should include the following packages (names may change
with time – the present version of PostgreSQL is 9.3):

  postgresql
  postgresql-client-9.3
  postgresql-server-dev-9.3
  libpq-dev
  libpq5
  postgresql-client-common
  postgresql-common
  postgresql-doc-9.3 (optional)

2. In the Users and Groups program for your system, add your personal userid to the postgres
group. This user id must not be root. This step is required in order to set locks.

On the main screen of Linux Mint, go to

System | Administration | Users and Groups
Click **Manage Groups**. Locate the `postgres` group and click it. Click **Properties** then put a check mark next to your userid. This makes you a member of the postgres group. Click Ok etc.

You need to reboot the system in order for this change to take effect.

3. In a terminal window, add the PostgreSQL bin directory to your path (for Ubuntu-like systems - check version for current number):

   ```sh
   PATH=$PATH:/usr/lib/postgresql/9.3/bin:
   export PATH
   ```

   (You probably want to add this to your `~/.bashrc` file so it will be done automatically on login). If you do not do this step, the commands below will not work unless they are prefixed by the appropriate directory paths. You should logout/login to your terminal again for this to be effective.

4. Become system root and give the user `postgres` a password; logout as root and login as `postgres` then add yourself as a postgres user. Now logout as `postgres`. The following is an example (in a terminal window):

   ```sh
   sudo passwd postgres
   su postgres
   createuser your_userid -s
   ```

   The -s option means superuser. In this text we assume that your PostgreSQL userid is the same as your Linux logon id.

5. Login (to Linux) as yourself and create the data base files and users:

   ```sh
   initdb -D data -W
   ```

   Replace `data` with the name of the directory to be created in which the PostgreSQL data base files will be stored. If you run this command from your home directory, your data directory will be `~/data`. If you want the PostgreSQL files located in another directory, modify this command as appropriate.

   You may include path information but you must have access rights to the directory in which you place the files. The -W option will result in being asked for a root password for PostgreSQL - this is the PostgreSQL root user, not the system root user. They may be the same, however, if you want but it is probably a good idea to use another password.

6. (Optional) Create additional users that can use this data base (userid is Linux user's login name).

   ```sh
   createuser userid
   ```

7. Create a database for mumps:

   ```sh
   createdb mumps
   ```

8. You may give your PostgreSQL userid a password if you want. Replace `password` by the password you have selected in the following:

   ```sh
   psql -d mumps
   ALTER USER your_postgreSQL_userid WITH PASSWORD 'password';
   
   Example dialog:
$ psql -d mumps
psql (9.3.5)
Type "help" for help.

mumps=# alter user mumps with password 'abc123';
ALTER ROLE
mumps=# \q

The `ALTER USER` command can be used to change the password of any PostgreSQL user because you gave yourself superuser authority in step 4.

Depending on your configuration, you may be asked for a password when connecting to the PostgreSQL server. This can be changed by altering the file `pg_hba.conf` located in `/etc/postgresql`.

In the most recent (9.3) version of PostgreSQL, passwords are required by default for local connections. In previous versions this was not always the case.

### 1.7.3.3 Mumps Build for PostgreSQL Resident Global Arrays

Mumps Build for mode B: using the configure command (as root) for Ubuntu, Mint, Debian, etc:

```
configure prefix=/usr 
--with-pgdb=/usr/include/postgresql 
--with-dname=mumps 
--with-pgsql-host=127.0.0.1 
--with-pgsql-user=mumps 
--with-pgsql-passwd=abc123
make
make install
```

Note: different versions of PostgreSQL have a habit of playing hide the files which may cause problems until I play hide and seek to correct the configuration files.

The resulting Mumps installation will store globals in the PostgreSQL server. The sever **must** be running and properly configured in order for the Mumps global array facility to function.

The distribution contains a BASH script file named `BuildMumpsWithGlobalsInPostgreSQL.script` which will perform the installation.

Once you have built a Mumps database in PostgreSQL, you may query it with general purpose SQL commands (such as `SELECT`, discussed elsewhere).

Note: it appears that some Synaptic package manager installs **may** incorrectly address the location of the Postgresql socket. If, upon starting mumps, you get a message that the connection could not be opened and to check the socket, you will need to correct an entry in the file:

```
/etc/postgresql/9.3/main/postgresql.conf
```

In this file, change the value for the entry `unix_socket_directory` to point to the directory in the error message (probably: `/var/run/postgresql`). This is a PostgreSQL issue, not a Mumps issue. You will need to restart the database after this or any other configuration changes:

```
/etc/init.d/postgresql restart
```

The following additional options are accepted by `configure`:

1. `--with-dname=[mumps]` This sets the name of the default database in which the Mumps globals will be stored. The name `mumps` will be used if this is omitted.
2. **--with-pgsql-host=[127.0.0.1]** This sets the IP number of the default machine on which the PostgreSQL server is running. If omitted, it defaults to 127.0.0.1.

### 1.7.3.4 Configuring the *listen_address*

By default, Mumps accesses the PostgreSQL server on the *current* machine. If you want to run Mumps programs on a different machine than the one running Mumps, you need to enable, on the *server* machine connections to its PostgreSQL server. The following are some brief instructions on how to permit a remote server to process Mumps requests. You should consult the PostgreSQL manuals for details which would be more appropriate to your application:

To accept connections, you should set (as root) the *listen_address* option in the file:

```
/etc/postgresql/9.3/main/postgresql.conf
```

to contain the IP numbers of the systems from which you are willing to accept connections. Note: the intermediate directory 9.3 in the above refers to the PostgreSQL release number. This will change with time.

For example:

```
listen_addresses = 'localhost,*'
```

The above, note the quotes, permits connections from all remote addresses. A restart is required:

```
/etc/init.d/postgresql restart
```

The connecting clients' IP numbers should be identified in the file *pg_hba.conf* found in the same directory. To enable a network connection, you should insert an line into this file. If you are using your IPv4 addresses, it should look something like:

```
host   all        all       10.42.0.0/16   trust
```

which means that the high order 16 bits of the IPV4 of the incoming request IP number must match 10.42 but the remaining 16 bits can be any value. The server will accept connections from any machine with the 10.42 prefix. The **trust** option means that a password will not be required from the connecting client. If you want the user to supply a password, use:

```
host   all        all       10.42.0.0/16   md5
```

There are other security options. Consult the PostgreSQL documentation.

You can test the connection to the server from a remote machine authorized to connect with the command:

```
psql -h 10.42.0.26 -d mumps
```

where the `-h` option specifies the remote host to connect to and the `-d medical` specifies the name of the database. This command assumes that the **trust** option was used and that the login userid of the user on the remote (client) system is the same as an authorized user on the remote (server) system. The server machine is at address 10.42.0.26. To exit from *psql*, type `
q` (backslash-q).

If you attempt to use mumps with PostgreSQL as the database from an account not recognized by PostgreSQL, you will receive the error messages:

```
*** Connection to database server failed in or near line 0
Error msg:
FATAL:  role "root" does not exist
Connection string=dbname=mumps
```

The PostgreSQL server can be started with SSL enabled by setting the parameter `ssl` to on in *postgresql.conf*.
**1.7.3.5 Performance Tuning**

By default, PostgreSQL is set for stringent data protection operation. This results in considerable disk activity to insure that data is never lost. However, many of these procedures slow the operation of the database during update to a considerable extent. They can, in many cases, be dispensed with with only minimal effects on database integrity.

The main configuration file is `postgresql.conf`. For a server started by the operating system, it will, by default, be in `/etc/postgresql/9.3/main`. If you started PostgreSQL manually with a command like:

```
postgres -D data_dir
```

where `data_dir` is the name of the directory you created with the `initdb` command as shown above, you will find the relevant `postgresql.conf` in `data_dir`.

Alter the settings as follows:

```
wal_level = minimal
fsync = off
synchronous_commit = off
full_page_writes = off
archive_mode = off
```

The result will be a considerable improvement in speed.

**1.7.3.6 Using PostgreSQL Resident Global Arrays from Mumps**

Each time you start Mumps you **must** establish a connection with the PostgreSQL server by means of the `sql/d` command. The prototype for the connection (found in `sysfunc.cpp.in`) is:

```
host=@remotehost@ dbname=@dbname@ user=@user@ password=@passwd@
```

The items between the at-signs are filled in by `configure` either from parameters you supply or defaults. This configuration text is used by default whenever you access the database.

The `sql/d` command may be entered in interactive mode or as part of a Mumps script file. If you want parameters other than the defaults, you must place them on the `sql/d` line and execute the line before any attempt to access the global arrays. The options available and their keywords are the same is given in the PostgreSQL documentation for the `PQconnectdb()` function.

The **first time** you use the `mumps` database, you **must** initialize it with the Mumps command:

```
sql/f
```

When you do this, you may see a message that the `mumps` tables do not exist. This can be ignored. A second running of the initialization command will not show the message. It is due to an SQL DROP statement on the mumps relation before database rebuild.

By default, Mumps will use localhost (127.0.0.1) to address the PostgreSQL server. If you want to use another host, use the Mumps command `sql/d`. This causes any current database connection to be closed. The remainder of the line consists of the new connection arguments, some of which may be of the &~exp~ format. If successful, `$t` will be true. This connection information will be used until changed or the Mumps client terminates. Examples:

```
sql/d dbname=mumps
sql/d host=abc.def.xyz.edu dbname=mumps
sql/d hostaddr=123.321.432.321 dbname=mumps
```
1.7.3.7 Use with Apache

When running through the CGI-BIN interface with the Apache web server, be sure your files and directories are not owned by root. Make them owned by Apache (user www-data in Linux Mint 13) and in the Apache group (also www-data). Apache's default cgi-bin directory is /usr/lib/cgi-bin. You will need to be root to add/modify files in this directory. Be sure to make Apache (www-data) an authorized PostgreSQL user with the createuser command (see above).

1.7.4 MySQL

1.7.4.1 MySQL Installation Options

The following are the configure options for MySQL. Several have default values.

1. --with-pgsql-user=user

The MySQL userid of the client Mumps program to be used when establishing a connection. May be set in the sql/d connection string. Default: mumps

2. --with-mysql-host=nbr

The IP number of the MySQL server. May be set in the sql/d connection string. Default: localhost

3. --with-mysqldb

Enables MySQL database storage of globals. MySQL is not enabled unless this option is specified. May not be set by sql/d connection string. Incompatible with the corresponding PostgreSQL enabling option.

4. --with-mysql

Enables MySQL access but globals are native. Not currently working.

5. --with-mysql-passwd=val

Specify, if needed, the MySQL user passwd. May be set in the sql/d connection string. Default: the empty string (no password).

6. --with-mysql-port=nbr

Port number to acces the MySQL server. Default: 0.

7. --with-mysql-socket=nbr

Socket through which to access the MySQL server. Default: NULL.

When installing MySQL you should create a user named mumps. By default, this user should have no password (you may change this in configure.ac or BuildWithMumpsInMysql.script).

1.7.4.2 Mumps Build for MySQL Resident Global Arrays

The script BuildMumpsWithGlobalsInMysql.script contains the code to build and configure Mumps to use a MySQL server. This script also sets the MySQL option for the compiler and the toolkit.

1.7.4.3 Configuring a Remote MySQL Server

See MySQL documentation.
1.7.4.4 Using MySQL Resident Global Arrays from Mumps

Same as for Postgresql with the exception of differences in the connection string used by sql/d.

1.7.4.5 Use with Apache

Similar to PostgreSQL. Consult Apache and MySQL documentation.

1.7.5 Use with Windows

You may build a native database version of Mumps in Windows using Cygwin for the native and MySQL database versions (not the native client-server). As of this writing, it has not been tested using PostgreSQL. An executable (mumps.exe) so built, my be copied to another directory and executed from a normal Windows command prompt if you copy several Cygwin DLLs to your system (\Windows) directory. These are:

cygcrypto-1.0.0.dll
cygcpre-1.dll
cygstdc++-6.dll
cygz.dll
cygmysqlclient-18.dll
cygssl-1.0.0.dll
cygwin1.dll
cygcc_s-1.dll

The first group is needed for all versions while the second group is also needed for MySSQL versions. These can be found in the Cygwin /bin directory.

If you build a MySQL version in Cygwin, even though it is running under Cygwin, Mumps will access the MySQL server running on Windows (or at a remote machine if you specify a remote IP number). You may run Mumps natively in a Windows command prompt box without Cygwin if you install the DLLs noted above in your Windows folder.

1.7.6 Normalization

If you place the global arrays in separate RDBMS tables, the issue of normalization becomes relevant. However, if all globals are stored in one table, normalization is not an issue.
2 Running a Mumps Program

2.1 Global Array Access

Note: if you are using the PosgreSQL option to store global arrays, you must inform the Mumps client of the name of the database and its location (sql/d command) and you must start the PostgreSQL server prior to attempting to access any global array.

If you are using single user mode native globals, you may access the globals now. If you are using the multi-user native globals, you must start the Mumps server daemon prior to attempting to access the globals.

2.2 Mumps CLI Interpreter

To run the interpreter in command line mode, type:

```
mumps
```

Any Mumps commands you enter will be executed immediately. To exit the interpreter, type H[alt].

In interactive mode, you will be presented with a prompt (>). Any Mumps command may be typed to this prompt for immediate execution (including a goto or do commands with a file name reference of a file to be loaded and executed).

The keyboard up arrow key may be used to cycle through and display commands previously entered during this session. A previously entered command may be reexecuted by using the keyboard up arrow key to locate and display the command and then typing <enter>. If, having begun cycling through previous commands, you decide not to re-execute a prior command, typing the <escape> key twice returns to the CLI to an empty prompt.

2.3 Mumps Programs (scripts)

Mumps programs are ASCII files which can be created by any editor and will normally have the following as their first line:

```
#!/usr/bin/mumps
```

The file extension .mps is preferred but not required. The Mumps source file must be made executable:

```
chmod u+x prog.mps
```

where prog.mps is the name of your mumps source file.

For example:

```
#!/usr/bin/mumps
for i=1:1:10 do
  .write "Hello World",i,!
halt
```

You may execute the program by typing prog.mps to your terminal prompt. The program above will write Hello World, followed by a number from 1 to 10, ten times.
3 Storing Mumps Globals in a Relational Database

As shown above, you may enable storage of global arrays in either the MySQL or PostgreSQL relational database systems. While access to globals is considerably slower than is the case with the native global array handler, the reliability and transaction processing capabilities offer many advantages.

In either MySQL or PostgreSQL the globals will be stored in a database named mumps. This may be changed with a configure option as shown in above. You must create this database before attempting to store globals in the database. The means to do this vary depending on which system you are using.

You must also create one or more users authorized to access the database. By default, the Mumps interpreter will attempt to login as user mumps. This may be changed either in the configure procedure or dynamically. Other options such as password, IP can be set. By default there is no password and localhost is assumed.

In the mumps database, the globals are stored in one or more tables. By default, all globals will be stored in a table named mumps. This table, by default (see configure), permits up to 10 levels of global array tree depth. You may, however, create additional tables.

When a global array is stored in a table, the first column is known as gbl to the RDBMS. There then follow a series of columns named a1, a2, a3, ... an. The number of columns is set when the table is created. By default, the mumps table has 11 columns: 10 for global array indices and one, a11, for any value stored at this reference. Thus, a table with $ztabsize of 10 actually has 12 columns.

You may have several tables in addition to mumps. The current table name is in the system variable $ztable and the maximum global array depth is in $ztabsize.

The $ztable variable may be used to switch to a different table during execution or, when creating a table, to specify the name of the table to be created.

$ztabsize is used to set the number columns when a table is created. Otherwise, it is used to inform as to the number of index columns in the current table.

Tables created by Mumps may be accessed outside of Mumps with standard SQL commands. Similarly, tables created outside of Mumps may be accessed by Mumps if an appropriate VIEW is created.

3.1 Mumps SQL commands

If relational database storage of globals is enabled, the following commands are available in the interpreter. If the native database is in use, these are ignored.

3.2 sql string

The remainder of the line is passed to the SQL server. The line should contain a valid SQL command and, normally, be terminated by a semi-colon. Any text of the form &~exp~ will result in exp being evaluated and the result replacing &~exp~. Mumps does not check the validity of the SQL command. The builtin Mumps variable $zsql will contain any messages or ‘ok’ if there were none. Do not use this command for SELECT queries.

This version of Mumps contains an added SQL command. There is no abbreviation for this command word. The remainder of the line on which it appears contains the arguments for the command. No other Mumps commands may appear on the line.

The SQL command controls access to the SQL server and may only be used if one of the SQL options (see above) has been enabled. The command has several forms. The form of the command is specified by a slash followed by a code letter or a code letter and a parameter.
3.3 sql/c SQL Disconnect

The sql/c command disconnects from the SQL server. This is normally done automatically when a program terminates if PostgreSQL or MySQL is being used to store the global array database. No other commands may appear on this line.

3.4 sql/d connect-string SQL Connection

Any current database connection is closed. The remainder of the line consists of new connection arguments, some of which may be of the &~exp~ format, the next time a connection is opened. If the connection is successful, $t will be true. This connection information will be used until changed or the Mumps client terminates.

The arguments differ slightly depending on whether you are using MySQL or PostgreSQL.

The configure procedure establishes defaults for database server logins. This command is needed only if these defaults are not adequate.

3.4.1 PostgreSQL Connections

By default, the connection is made to the database mumps on local host.

The string is the information needed to connect to the PostgreSQL server. At a minimum, it should include the name of the database being connected to:

dbname=medical

Other options include the host, host address, user, password, etc. ude default. For a local server, you probably only need the dbname= parameter and be sure that (1) you are running as a known PostgreSQL user and (2) that you have read/write privs in the directory you are running in.

Examples:

sql/d dbname=medical

sql/d host=abc.def.xyz.edu dbname=medical

sql/d hostaddr=123.321.432.321 dbname=medical

sql/d user=joe password=abc123 dbname=medical host=abc.def.xyz.edu

set x="dbname=medical"

sql/d &~x~

3.4.2 MySQL Connections

For MySQL the connection string consists of one or more of the following. If omitted, the value specified in configure will be used. Note: no embedded blanks are permitted. Note spelling of passwd.

user=mysqlUserId
host=IPNumber
passwd=value
dbname=DBName
port=value
socket=value

Example:

sql/d user=joe passwd=abc123 dbname=mumps host=abc.def.xyz.edu port=123

3.5 sql/f Format SQL Table
The form: \texttt{sql\slash f} instructs the relational database backend to initialize a table for global array use and delete any previous contents.

If the command contains no arguments, the default table (\texttt{mumps} or such other name determined by the \texttt{configure} procedure) will be initialized and formatted.

The command may contain two arguments: a relational table name and the number of columns. The arguments must not contain blanks and are separated by blanks. The first argument is the name of the relational table to be created to store Mumps globals. It's name must conform to the naming requirements for a Mumps variable (no underscore characters, for example). It should be lower case as some relational database systems do not differentiate between upper and lower case table names. The numeric argument may range between 1 and 20. It is the number of columns in the database exclusive of a column containing the global array name and a column containing any stored data. It represents the maximum number of global array indices in any given global array reference in this table. Examples:

\begin{verbatim}
  sql\slash f labs 3
  sql\slash f meds 5
\end{verbatim}

If either argument is present, both must be present. Variables or expressions are not permitted. In the case of the first example, the maximum depth of a global would be 3 and, in the case of the second, 5.

If no arguments appear on this line the current table name in \texttt{$ztable$} and the current number of columns in \texttt{$ztabsize$} will be used to create/initialize the table. By default, the initial value of \texttt{$ztable$} is \texttt{mumps} and the initial value of \texttt{$ztabsize$} is 10 (these can be changed in \texttt{configure}).

### 3.6 Added builtin SQL Variables

1. \texttt{$zsql$} Returns the SQL server error message for the most recent command or 'ok.'

2. \texttt{$zsqlOpen$} Returns true if a connection to the SQL server is open, false otherwise.

   a) \texttt{$znative$} returns true if globals are being stored in the native global array
   
   b) \texttt{$zmysql$} returns true if globals are being stored in MySQL
   
   c) \texttt{$zpostgres$} returns true if globals are being stored in PostgreSQL

3. \texttt{$ztable$} returns a comma separated string. The portion prior to the comma is the current RDBMS table in which the Mumps globals are stored. The part after the comma is the maximum number of indices permitted in the table (same as \texttt{$ztabsize$}).

   \texttt{$ztable$} may be set. If it is set immediately prior to an \texttt{sql\slash f} command, it is the name of the table to be created and/or initialized which now becomes the default table.

   If \texttt{$ztable$} is set to the name of a table which exists, global array reference will be made to this table until \texttt{$ztable$} is changed. When \texttt{$ztable$} is changed to the name of an existing table, \texttt{$ztabsize$} is updated to the value for the new current table.

4. \texttt{$ztabsize$} returns the number of RDBMS columns available for global array indexes. May be set immediately prior to an \texttt{sql\slash f} command in which case the value in \texttt{$ztabsize$} will be used to set the number of columns (range: 1 to 20).

### 3.7 PostgreSQL Performance Tuning

PostgreSQL has many internal tuning parameters that are not covered here. See the PostgreSQL documentation.

However, in cases where speed is important, a series of inserts into the database that are done as one transaction is faster than individual transactions (the default. For example:
One line 3 the database is cleared and initialized. On line 8 the SQL command disables the server's waits for the transaction's records to be flushed to permanent storage before returning a success indication to the client. This causes the inserts to proceed very much faster but at some risk of data loss (but not data corruption) should the system fail during updates.

Line 9 starts a transaction and line 19 commits the transaction which finalizes the values inserted in the intervening lines.

The file /etc/postgresql/9.3/main/postgresql.conf (this is the address on the current version - later versions may be different with respect to the version number) contains a number of settable parameters.

### 3.8 Examples

#### 3.8.1 DBMS Based Global Arrays

##### 3.8.1.1 Using a Single RDBMS Table

The following examples were done using the MySQL server. There are some minor differences in when using PostgreSQL.

First, create the mumps table in the server:

```bash
#!/usr/bin/mumps
sql/f
sql/d host=10.42.0.26 dbname=mumps
set k=0
sql SET LOCAL synchronous_commit TO OFF;
sql begin transaction;
for i=1:1:100 do
  . s k=k+1
  . s ^a(i)=k
for j=1:1:100 do
  .. set k=k+1
  .. set ^a(i,j)=k
for m=1:1:10 do
  ... s k=k+1
  ... s ^a(i,j,m)=k
sql commit
halt
```

The above creates and initializes the table in which the globals will be stored.

Next we load several global arrays with randomly created medical data. Note: all data, names and places were generated by a random number generator. You aunt Tilly's data isn't here.

Note also that the procedures shown in this section are essentially the same as in the next with the only difference being that in this section, all the globals are stored in one table while in the next, they are stored in separate tables.

The script file to load the database is shown below. The input files (labs, labtab, probtab, problems, medtab, medadmin, patientid and vitals) are the same as shown in section 3.8.1.2 below.

```bash
#!/bin/bash
loadlabs1.mps < labs
loadlabtab1.mps < labtab
loadprotab1.mps < probtab
```
The programs are essentially similar to those in section 3.8.1.2 but without the creation of separate tables. They are:

loadlabs1.mps

#!/usr/bin/mumps
for i=1:1:1000 do
  . read a
  . if '$test break
  . set a=$znoblanks(a)
  . s ptid=$p(a,"#",1)
  . s date=$p(a,"#",2)
  . s time=$p(a,"#",3)
  . s test=$p(a,"#",4)
  . s rslt=$p(a,"#",5)
  . w ".
  . set ^labs(ptid,test,date,time)=rslt
  w !

loadlabtab1.mps

#!/usr/bin/mumps
for i=1:1:1000 do
  . read a
  . if '$test break
  . s name=$p(a,"#",2)
  . set a=$znoblanks(a)
  . s test=$p(a,"#",1)
  . s low=$p(a,"#",3)
  . s high=$p(a,"#",4)
  . set "labtab(test,name,low)=high
  . w ".
  write !

loadprobtab1.mps

#!/usr/bin/mumps
for i=1:1:1000 do
  . read a
  . if '$test break
  . set a=$znoblanks(a)
  . s icd=$p(a,"#",1)
  . s dx=$p(a,"#",2)
  . w ".
  . set "probtab(icd)=dx
  w !

loadprobs1.mps

#!/usr/bin/mumps
for i=1:1:1000 do
  . read a
  . if '$test break
  . set a=$znoblanks(a)
  . s ptid=$p(a,"#",1)
  . s onset=$p(a,"#",2)
  . s resolv=$p(a,"#",3)
  . s dxphys=$p(a,"#",4)
  . w ".
  . set "problems(ptid,onset,resolv)=dxphys
  w !
#!/usr/bin/mumps

for i=1:1:1000 do
 . read a
 . if 'test break
 . s medname=$p(a,"#",2)
 . set a=$znoblanks(a)
 . s medcode=$p(a,"#",1)
 . s lowdose=$p(a,"#",3)
 . s highdose=$p(a,"#",4)
 . w ""
 . set ^medtab(medcode,lowdose,highdose)=medname
w !

w !
loadmedadmin1.mps

#!/usr/bin/mumps

for i=1:1:1000 do
 . read a
 . if 'test break
 . set a=$znoblanks(a)
 . s ptid=$p(a,"#",1)
 . s medcode=$p(a,"#",2)
 . s dose=$p(a,"#",3)
 . s date=$p(a,"#",4)
 . s time=$p(a,"#",4)
 . w ""
 . set ^medadmin(ptid,medcode,dose,date)=time
w !

w !
loadpatientid1.mps

#!/usr/bin/mumps

for i=1:1:1000 do
 . read a
 . if 'test break
 . s name=$p(a,"#",4)
 . set a=$znoblanks(a)
 . s ptid=$p(a,"#",1)
 . s date=$p(a,"#",2)
 . s time=$p(a,"#",3)
 . s prefix=$p(a,"#",5)
 . s suffix=$p(a,"#",6)
 . s street=$p(a,"#",7)
 . s citystate=$p(a,"#",8)
 . s zip=$p(a,"#",9)
 . s tel=$p(a,"#",10)
 . s insurercode=$p(a,"#",11)
 . set "patientid(ptid,date,time,street,citystate,zip,tel,insurercode)=""
 . set "ptname(ptid)=name"
 . write "."
write !

w !
loadvitals1.mps

#!/usr/bin/mumps

for i=1:1:1000 do
 . read a
 . if 'test break
 . set a=$znoblanks(a)
 . s ptid=$p(a,"#",1)
 . s date=$p(a,"#",2)
 . s time=$p(a,"#",3)
 . s sys=$p(a,"#",4)
 . s dia=$p(a,"#",5)
 . s ptid=$p(a,"#",1)
 . s date=$p(a,"#",2)
 . s time=$p(a,"#",3)
 . s sys=$p(a,"#",4)
 . s dia=$p(a,"#",5)
The results in the table `mumps` in the MySQL server using the command look like:
### drop view if exists

```sql
drop view if exists med_admin;
drop view if exists med_table;
drop view if exists problem_list;
drop view if exists problem_table;
drop view if exists lab_rslts;
drop view if exists lab_table;
```

### Constants and Vars

<table>
<thead>
<tr>
<th>gbl</th>
<th>a1</th>
<th>a2</th>
<th>a3</th>
<th>a4</th>
<th>a5</th>
<th>a6</th>
<th>a7</th>
<th>a8</th>
<th>a9</th>
<th>a10</th>
<th>a11</th>
</tr>
</thead>
<tbody>
<tr>
<td>probtab</td>
<td>(308 rows)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>probtab</td>
<td>001</td>
<td>Cholera</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>probtab</td>
<td>002</td>
<td>Typhoid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>probtab</td>
<td>003</td>
<td>Other</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>probtab</td>
<td>004</td>
<td>Shigellosis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>probtab</td>
<td>005</td>
<td>Other</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>probtab</td>
<td>006</td>
<td>Amebiasis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>probtab</td>
<td>007</td>
<td>Intestinal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>probtab</td>
<td>008</td>
<td>Ill-defined</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Note

The single `mumps` table is one continuous table. The excerpts shown above were the results of commands such as:

```sql
select * from mumps where gbl='var' limit 10;
```

where `var` was the several global array names.

The single `mumps` table, however, can be mapped into separate views as shown in the following:

```sql
drop view if exists lab_table;
drop view if exists lab_rslts;
drop view if exists problem_table;
drop view if exists problem_list;
drop view if exists med_table;
drop view if exists med_admin;
```
select * from lab_table limit 10;
select * from lab_rslts limit 10;
select * from problem_table limit 10;
select * from problem_list limit 10;
select * from med_table limit 10;
select * from med_admin limit 10;
select * from patient_id limit 10;
select * from pt_name limit 10;
select * from vital_signs limit 10;

select patient_id.ptid, pt_name.name, lab_table.test_name
from pt_name, patient_id, lab_table, lab_rslts
where patient_id.ptid = lab_rslts.ptid and pt_name.ptid = patient_id.ptid and lab_rslts.test = lab_table.test_code limit 10;

yields:

<table>
<thead>
<tr>
<th>test_code</th>
<th>test_name</th>
<th>low</th>
<th>high</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5-HIAA</td>
<td>6</td>
<td>76</td>
</tr>
<tr>
<td>10</td>
<td>Halothane (Anaesthetic)</td>
<td>17</td>
<td>56</td>
</tr>
<tr>
<td>100</td>
<td>MTX</td>
<td>9</td>
<td>58</td>
</tr>
<tr>
<td>101</td>
<td>Mucopolysaccharides</td>
<td>49</td>
<td>50</td>
</tr>
<tr>
<td>102</td>
<td>Multimer Analysis</td>
<td>35</td>
<td>86</td>
</tr>
<tr>
<td>103</td>
<td>T3, Total</td>
<td>8</td>
<td>87</td>
</tr>
<tr>
<td>104</td>
<td>T4, Free</td>
<td>32</td>
<td>63</td>
</tr>
<tr>
<td>105</td>
<td>T4, Free by Equilibrium Dialysis</td>
<td>23</td>
<td>77</td>
</tr>
<tr>
<td>106</td>
<td>T4/T8 (All Requests)</td>
<td>44</td>
<td>80</td>
</tr>
</tbody>
</table>

10 rows in set (0.00 sec)
<table>
<thead>
<tr>
<th>icd</th>
<th>dx</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>Cholera</td>
</tr>
<tr>
<td>002</td>
<td>Typhoid</td>
</tr>
<tr>
<td>003</td>
<td>Other</td>
</tr>
<tr>
<td>004</td>
<td>Shigellosis</td>
</tr>
<tr>
<td>005</td>
<td>Other</td>
</tr>
<tr>
<td>006</td>
<td>Amebiasis</td>
</tr>
<tr>
<td>007</td>
<td>Other</td>
</tr>
<tr>
<td>008</td>
<td>Intestinal</td>
</tr>
<tr>
<td>009</td>
<td>Ill-defined</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ptid</th>
<th>icd</th>
<th>onset</th>
<th>resolved</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000000</td>
<td>022</td>
<td>1995-03-23</td>
<td>1969-08-18</td>
</tr>
<tr>
<td>1001000</td>
<td>694</td>
<td>1950-09-13</td>
<td>1951-04-07</td>
</tr>
<tr>
<td>1002000</td>
<td>227</td>
<td>1994-10-28</td>
<td>1981-01-20</td>
</tr>
<tr>
<td>1003000</td>
<td>049</td>
<td>1978-07-04</td>
<td>1955-05-06</td>
</tr>
<tr>
<td>1004000</td>
<td>116</td>
<td>1972-08-05</td>
<td>1959-09-25</td>
</tr>
<tr>
<td>1005000</td>
<td>150</td>
<td>1976-07-27</td>
<td>1982-03-11</td>
</tr>
<tr>
<td>1006000</td>
<td>124</td>
<td>1983-10-15</td>
<td>1965-07-03</td>
</tr>
<tr>
<td>1007000</td>
<td>693</td>
<td>1983-10-09</td>
<td>1985-08-19</td>
</tr>
<tr>
<td>1008000</td>
<td>098</td>
<td>1973-05-02</td>
<td>1957-07-07</td>
</tr>
<tr>
<td>1009000</td>
<td>090</td>
<td>1984-01-05</td>
<td>1990-05-06</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>medcode</th>
<th>lowdose</th>
<th>highdose</th>
<th>med_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>16</td>
<td>44</td>
<td>ACEPROMAZINE</td>
</tr>
<tr>
<td>10001</td>
<td>18</td>
<td>66</td>
<td>ACTIGALL</td>
</tr>
<tr>
<td>10002</td>
<td>17</td>
<td>63</td>
<td>ADEQUAN</td>
</tr>
<tr>
<td>10003</td>
<td>10</td>
<td>77</td>
<td>ADRIAMYCIN</td>
</tr>
<tr>
<td>10004</td>
<td>14</td>
<td>92</td>
<td>ADVANTAGE</td>
</tr>
<tr>
<td>10005</td>
<td>18</td>
<td>85</td>
<td>ALBON</td>
</tr>
<tr>
<td>10006</td>
<td>4</td>
<td>63</td>
<td>ALLOPURINOL</td>
</tr>
<tr>
<td>10007</td>
<td>8</td>
<td>90</td>
<td>AMITRIPTYLINE</td>
</tr>
<tr>
<td>10008</td>
<td>4</td>
<td>99</td>
<td>AMLODIPINE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ptid</th>
<th>medcode</th>
<th>dose</th>
<th>date</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000000</td>
<td>22</td>
<td>34</td>
<td>1970-01-07</td>
<td>22:07:00</td>
</tr>
<tr>
<td>1001000</td>
<td>43</td>
<td>0</td>
<td>1975-02-02</td>
<td>1975-02-02</td>
</tr>
<tr>
<td>1002000</td>
<td>61</td>
<td>70</td>
<td>1991-03-30</td>
<td>1991-03-30</td>
</tr>
<tr>
<td>1004000</td>
<td>126</td>
<td>182</td>
<td>1996-06-11</td>
<td>1996-06-11</td>
</tr>
<tr>
<td>1005000</td>
<td>58</td>
<td>52</td>
<td>1984-10-01</td>
<td>1984-10-01</td>
</tr>
<tr>
<td>1006000</td>
<td>52</td>
<td>182</td>
<td>1972-03-25</td>
<td>1972-03-25</td>
</tr>
<tr>
<td>1007000</td>
<td>203</td>
<td>132</td>
<td>1988-11-26</td>
<td>1988-11-26</td>
</tr>
<tr>
<td>1008000</td>
<td>238</td>
<td>38</td>
<td>1954-09-03</td>
<td>1954-09-03</td>
</tr>
<tr>
<td>1009000</td>
<td>183</td>
<td>113</td>
<td>1956-07-03</td>
<td>1956-07-03</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ptid</th>
<th>date</th>
<th>time</th>
<th>street</th>
<th>citystate</th>
<th>zip</th>
<th>tel</th>
<th>insurer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000000</td>
<td>1991-01-16</td>
<td>22:07:00</td>
<td>237GreenLane</td>
<td>Chesterville,Ontario</td>
<td>73960</td>
<td>(598)481-2785</td>
<td>999</td>
</tr>
<tr>
<td>1001000</td>
<td>1977-08-08</td>
<td>08:45:00</td>
<td>179BluffAve.</td>
<td>Selkirk,Manitoba</td>
<td>17920</td>
<td>(232)431-9470</td>
<td>140</td>
</tr>
<tr>
<td>1002000</td>
<td>1981-09-12</td>
<td>09:12:00</td>
<td>845CrestLane</td>
<td>NewGlasgow,NovaScotia</td>
<td>3162</td>
<td>(978)754-1751</td>
<td>160</td>
</tr>
<tr>
<td>1003000</td>
<td>1968-08-05</td>
<td>07:34:00</td>
<td>385ChaseAve.</td>
<td>NotreDameDeLourdes,Quebec</td>
<td>16661</td>
<td>(888)776-2020</td>
<td>140</td>
</tr>
<tr>
<td>1004000</td>
<td>1998-03-01</td>
<td>04:41:00</td>
<td>353TreasureLaneApt72</td>
<td>Bath,NewBrunswick</td>
<td>27103</td>
<td>(994)186-1738</td>
<td>160</td>
</tr>
<tr>
<td>1005000</td>
<td>1998-03-24</td>
<td>11:27:00</td>
<td>430HarvestLaneApt16</td>
<td>Chesterville,Ontario</td>
<td>13687</td>
<td>(798)995-2598</td>
<td>110</td>
</tr>
<tr>
<td>1006000</td>
<td>1970-07-16</td>
<td>06:26:00</td>
<td>984LagoonAve.</td>
<td>PortElgin,Ontario</td>
<td>87985</td>
<td>(602)837-4680</td>
<td>160</td>
</tr>
<tr>
<td>1007000</td>
<td>1974-09-26</td>
<td>02:10:00</td>
<td>420ViewRd.</td>
<td>MissouriCity,Texas</td>
<td>47972</td>
<td>(860)853-6684</td>
<td>120</td>
</tr>
<tr>
<td>1008000</td>
<td>1987-10-09</td>
<td>12:28:00</td>
<td>926KnowlAve.</td>
<td>Stratford,PrinceEdwardIsland</td>
<td>4786</td>
<td>(536)231-2440</td>
<td>999</td>
</tr>
<tr>
<td>1009000</td>
<td>1972-04-23</td>
<td>20:04:00</td>
<td>420ViewRd.</td>
<td>Dorchester,Ontario</td>
<td>77425</td>
<td>(569)415-8753</td>
<td>130</td>
</tr>
<tr>
<td>ptid</td>
<td>name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000000</td>
<td>Kemp Widaman</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1001000</td>
<td>Karolyn Hindman</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1002000</td>
<td>Gussie Bickerson</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1003000</td>
<td>Lenox Giesen</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1004000</td>
<td>Leta Mcmichaels</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1005000</td>
<td>Laureen Tanner</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1006000</td>
<td>Hamilton Reade</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1007000</td>
<td>Netta Garratt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1008000</td>
<td>Rebekah Wynter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1009000</td>
<td>Kemp Widaman</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

10 rows in set (0.00 sec)

<table>
<thead>
<tr>
<th>ptid</th>
<th>date</th>
<th>time</th>
<th>sys</th>
<th>dia</th>
<th>resp</th>
<th>temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000000</td>
<td>1981-02-08</td>
<td>08:23:00</td>
<td>73</td>
<td>90</td>
<td>50</td>
<td>11</td>
</tr>
<tr>
<td>1001000</td>
<td>1980-02-27</td>
<td>17:38:00</td>
<td>59</td>
<td>201</td>
<td>74</td>
<td>26</td>
</tr>
<tr>
<td>1002000</td>
<td>1983-01-26</td>
<td>02:39:00</td>
<td>70</td>
<td>145</td>
<td>105</td>
<td>13</td>
</tr>
<tr>
<td>1003000</td>
<td>1983-09-17</td>
<td>14:26:00</td>
<td>82</td>
<td>190</td>
<td>59</td>
<td>16</td>
</tr>
<tr>
<td>1004000</td>
<td>1974-05-22</td>
<td>19:24:00</td>
<td>110</td>
<td>214</td>
<td>63</td>
<td>23</td>
</tr>
<tr>
<td>1005000</td>
<td>1968-03-20</td>
<td>16:24:00</td>
<td>95</td>
<td>129</td>
<td>61</td>
<td>26</td>
</tr>
<tr>
<td>1006000</td>
<td>1967-09-23</td>
<td>09:14:00</td>
<td>64</td>
<td>76</td>
<td>80</td>
<td>26</td>
</tr>
<tr>
<td>1007000</td>
<td>1972-04-15</td>
<td>01:22:00</td>
<td>92</td>
<td>127</td>
<td>82</td>
<td>26</td>
</tr>
<tr>
<td>1008000</td>
<td>1982-11-29</td>
<td>13:11:00</td>
<td>72</td>
<td>148</td>
<td>52</td>
<td>12</td>
</tr>
<tr>
<td>1009000</td>
<td>1988-09-13</td>
<td>12:27:00</td>
<td>87</td>
<td>154</td>
<td>53</td>
<td>14</td>
</tr>
</tbody>
</table>

10 rows in set (0.00 sec)

<table>
<thead>
<tr>
<th>ptid</th>
<th>name</th>
<th>test_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000000</td>
<td>Kemp Widaman</td>
<td>Tofranil Water Deprivation Test</td>
</tr>
<tr>
<td>1001000</td>
<td>Karolyn Hindman</td>
<td>Chlamydia Test</td>
</tr>
<tr>
<td>1002000</td>
<td>Gussie Bickerson</td>
<td>Alcohol Screen - Blood</td>
</tr>
<tr>
<td>1003000</td>
<td>Lenox Giesen</td>
<td>A2 Hemoglobin</td>
</tr>
<tr>
<td>1004000</td>
<td>Leta Mcmichaels</td>
<td>5-HIAA</td>
</tr>
<tr>
<td>1005000</td>
<td>Laureen Tanner</td>
<td>Hemoglobin</td>
</tr>
<tr>
<td>1006000</td>
<td>Hamilton Reade</td>
<td>Molecular Diagnostic Tests</td>
</tr>
<tr>
<td>1007000</td>
<td>Netta Garratt</td>
<td>Rickettsiae Serology</td>
</tr>
<tr>
<td>1008000</td>
<td>Rebekah Wynter</td>
<td>Ceruloplasmin</td>
</tr>
<tr>
<td>1009000</td>
<td>Kemp Widaman</td>
<td>Enteric Pathogen Stool Culture</td>
</tr>
</tbody>
</table>

10 rows in set (0.00 sec)

3.8.1.2 Using Multiple RDBMS Tables

The following are a set of examples that show how to use RDBMS based global arrays stored in multiple RDBMS tables and how to access other RDBMS tables from within Mumps.

First we create a database from randomly generated data. Note: the names, addresses and tests in these examples are randomly generated. They are not real people or real data.

To duplicate these, first build Mumps for either MySQL or PostgreSQL, install the MySQL or PostgreSQL software if needed, create a user named mumps (no password) and create a database likewise known as mumps.

```bash
#!/bin/bash
loadlabs.mps < labs
loadlabtab.mps < labtab
loadprotab.mps < probtab
loadprobs.mps < problems
loadmedtab.mps < medtab
loadmedadmin.mps < medadmin
loadpatientid.mps < patientid
loadvitals.mps < vitals

loads.mps BASH Script to Generate the Global Array Database
```
Each of the above loads an aspect of the database into the RDBMS resident globals. The input to each step is a file containing randomly generated data. Samples are given below.

In each Mumps program a separate RDBMS table is created to hold each global array. Alternatively, all the global arrays could have been stored in the default table named `mumps`. However, placing the globals in separate tables makes them more easily accessed with ordinary SQL commands as shown below.

Each Mumps program sets `$ztable` to the name of the RDBMS table to be used and `$ztabsize` is set to reflect the number of columns that will be needed for the global array indices. For example, in `loadlabs.mps`, the name of the table is `labs` and the number of columns is set to 4. In each program the `sql/f` command is executed. It creates (or re-creates) and initializes the table with the correct number of columns.

Each program then begins reading input lines (up to 1,000) and breaks each down into separate parts. Extra blanks are removed as necessary. Finally, the input line contents are contained in a set of Mumps variables. These are used as indices and stored values in storing a global array. Note: while the globals arrays have the same name as the tables into which they are stored, this is not required. It does, however, make things easier to deal with. Each program outputs dots to show its progress. The program `loadpatientid.mps` creates two global arrays but in separate tables.

```mumps
#!/usr/bin/mumps
set $ztable="labs"
set $ztabsize=4
sql/f
for i=1:1:1000 do
. read a
. if "$test break
. set a=$znoblanks(a)
. s ptid=$p(a,"#",1)
. s date=$p(a,"#",2)
. s time=$p(a,"#",3)
. s test=$p(a,"#",4)
. s rslt=$p(a,"#",5)
. w "."
. set "labs(ptid,test,date,time)=rslt
w !
loadlabs.mps
```

Sample of the file `labs`

```
101000  #1978-06-06#03:30:00#411#58
102000  #1967-07-21#02:51:00#394#28
103000  #1989-01-07#04:12:22:44:00#394#20
104000  #1982-11-10#19:22:00#149#1
105000  #1955-06-21#22:44:00#180#96
106000  #1994-02-08#24:00#39#91
107000  #1960-09-13#38:00#320#89
108000  #1968-02-19#08:17:00#298#4
109000  #1953-03-17#11:57:00#226#31
110000  #1975-09-28#15:00#282#75
111000  #1986-06-03#18:00#292#37
112000  #1966-01-20#17:15:00#271#40
113000  #1997-04-17#20:39:00#282#75
```

```mumps
#!/usr/bin/mumps
set $ztable="labtab"
set $ztabsize=3
sql/f
for i=1:1:1000 do
. read a
. if "$test break
. s name=$p(a,"#",2)
```

Sample of the file `labtab`
. set a=$znoblanks(a)
. s test=$p(a, ",", 1)
. s low=$p(a, ",", 3)
. s high=$p(a, ",", 4)
. set "labtab(test, name, low)=high
. w "."

write !

loadlabtab.mps

1#5-HIAA#6#76
2#5-Hydroxyindoleacetic Acid#8#83
3#5-Hydroxytryptamine#2#55
4#17-Hydroxyprogesterone#3#74
5#18-Hydroxycorticosterone#2#86
6#25-Hydroxy Vitamin D#3#52
7#Haemophilus ducreyi#32#62
8#Haldol#31#62
9#Haloperidol#12#78
10#Halothane (Anaesthetic)#17#56
11#Ham Test#25#60
12#Hantavirus Serology#34#58
13#Haptoglobin#30#52
14#HBeAg (Hepatitis B Antigen)#14#73
15#HBsAg#25#89
16#HCG#20#71
17#HDL Cholesterol#38#50

Sample of file labtab

#!/usr/bin/mumps
set $ztable="probtab"
set $ztabsize=1
sql/f
for i=1:1:1000 do
. read a
. if 'test break
. set a=$znoblanks(a)
. s icd=$p(a, ",", 1)
. s dx=$p(a, ",", 2)
. w "."
. set "probtab(icd)=dx
. w !

loadprotab.mps

001 #Cholera
002 #Typhoid
003 #Other
004 #Shigellosis
005 #Other
006 #Amebiasis
007 #Other
008 #Intestinal
009 #Ill-defined
010 #Primary
011 #Pulmonary
012 #Other
013 #Tuberculosis
014 #Tuberculosis
015 #Tuberculosis
016 #Tuberculosis
017 #Tuberculosis
018 #Miliary

Sample of file probtab

#!/usr/bin/mumps
set $ztable="problems"
set $ztabsize=3
sql/f

for i=1:1:1000 do
  . read a
  . if 'test break
  . set a=$znoblanks(a)
  . s ptid=$p(a,"#",1)
  . s onset=$p(a,"#",2)
  . s resolv=$p(a,"#",3)
  . s dxphys=$p(a,"#",4)
  . w "."
  . set "problems(ptid,onset,resolv)=dxphys"

w !

loadprobs.mps

Sample of file problems

#!/usr/bin/mumps

set $ztable="medtab"
set $ztabsize=3
sql/f

for i=1:1:1000 do
  . read a
  . if 'test break
  . s medname=$p(a,"#",2)
  . set a=$znoblanks(a)
  . s medcode=$p(a,"#",1)
  . s lowdose=$p(a,"#",3)
  . s highdose=$p(a,"#",4)
  . w "."
  . set "medtab(medcode,lowdose,highdose)=medname"

w !

loadmedtab.mps

Sample of file problems

10000  #117  #1960-03-16#1989-02-20#909
102000  #494  #1955-06-06#1984-09-07#29
103000  #065  #1971-11-18#1995-01-20#421
104000  #048  #1988-01-05#1980-05-11#702
105000  #084  #1966-11-02#1995-11-29#847
106000  #504  #1972-02-15#1973-09-29#792
107000  #490  #1963-01-06#1957-02-28#240
108000  #086  #1982-03-08#1988-11-12#505
109000  #132  #1957-07-06#1993-09-29#382
110000  #686  #1974-05-01#1989-07-09#782
111000  #774  #1992-11-30#1980-07-16#188
112000  #221  #1959-02-24#1970-01-12#791
113000  #159  #1979-10-25#1960-03-06#876
114000  #079  #1955-04-21#1971-06-28#365
115000  #151  #1969-05-26#1997-04-01#950
116000  #190  #1989-05-11#1965-08-26#635
117000  #104  #1961-06-25#1960-01-10#800
118000  #775  #1954-09-15#1956-05-08#604
#!/usr/bin/mumps
set $ztable="medadmin"
set $ztabsize=4
sql/f
for i=1:1:1000 do
  . read a
  . if '$test break
  . set a=$znoblanks(a)
  . s ptid=$p(a,"#",1)
  . s medcode=$p(a,"#",2)
  . s dose=$p(a,"#",3)
  . s date=$p(a,"#",4)
  . s time=$p(a,"#",5)
  . w "."
  . set "medadmin(ptid,medcode,dose,date)=time
  w !
loadmedadmin.mps

Sample of file medadmin

#!/usr/bin/mumps
set $ztable="patientid"
set $ztabsize=8
sql/f
set $ztable="ptname"
set $ztabsize=1
sql/f
for i=1:1:1000 do
  . read a
  . if '$test break
  . set a=$znoblanks(a)
  . s name=$p(a,"#",4)
  . s ptid=$p(a,"#",1)
  . s date=$p(a,"#",2)
  . s time=$p(a,"#",3)
  . s prefix=$p(a,"#",5)
  . s suffix=$p(a,"#",6)
  . s street=$p(a,"#",7)
  . s citystate=$p(a,"#",8)
  . s zip=$p(a,"#",9)
  . s tel=$p(a,"#",10)
  . s insurercode=$p(a,"#",11)
  . set $ztable="patientid"
. set "patientid(ptid,date,time,street,citystate,zip,tel,insurercode)="
. set $ztable="ptname"
. set $ztable="ptname"=name
 . write ".
 . write !

loadpatientid.mps

101000 #1951-02-1415:27:00#Cynthia Adcock#Mr.#Sr.#956 Park Lane #Russell, Manitoba#365644#165 564-3155 $180
102000 #1961-06-07#1951-02-14#0004 Cherise Allington#Mr.#Sr.#969 Place Ave. Apt 56#Stratford, Quebec#407638(971) 672-2871 $999
103000 #1964-02-0719:06:00#Fortune Baxter#Mr.#Sr.#43 Drive Lane #Idaho Springs, Colorado#71384#118 638-3065 $130
104000 #1966-10-1619:40:00#Jenny Ashbaugh#Mr.#Sr.#Round Lane Apt #6Virginia City, Nevada#79348#173 673-2722 $100
105000 #1950-04-12#1950-04-12#0100
106000 #1968-11-07#1968-11-17#0300#1100
107000 #1966-09-07#1966-09-09#0200
108000 #1991-08-04#1991-08-04#140
109000 #1956-04-05#1956-04-05#100
110000 #1961-10#1962000#170
111000 #1961-11#1961-11#180
112000 #1987-11-18#1987-11-18#110
113000 #1973-01-03#1973-01-03#160

Sample of file patientid

#!/usr/bin/mumps
set $ztable="vitals"
set $ztabsize=7
sql/f
for i=1:1:1000 do
 . read a
 . if '$test break
 . set a=$znoblanks(a)
 . s ptid=$p(a,"#",1)
 . s date=$p(a,"#",2)
 . s time=$p(a,"#",3)
 . s sys=$p(a,"#",4)
 . s dia=$p(a,"#",5)
 . s resp=$p(a,"#",6)
 . s temp=$p(a,"#",7)
 . w "."
 . set "vitals(ptid,date,time,sys,dia,resp)=temp
 . w !

loadvitals.mps

Sample of file vitals

#!/bin/bash
for i in 1:1:10000 do
 . echo "labtab"
 . mysql -u mumps -D mumps --execute="select * from labtab limit 10;"
 . echo "labs"
 . mysql -u mumps -D mumps --execute="select * from labs limit 10;"

After loads.mps has executed each of the Mumps programs, the database can be displayed with:

#!/bin/bash
```bash
```
Which, when run, yields:

### labtab
```
<table>
<thead>
<tr>
<th>gbl</th>
<th>a1</th>
<th>a2</th>
<th>a3</th>
<th>a4</th>
</tr>
</thead>
<tbody>
<tr>
<td>labtab</td>
<td>1</td>
<td>5-HIAA</td>
<td>6</td>
<td>76</td>
</tr>
<tr>
<td>labtab</td>
<td>2</td>
<td>5-Hydroxyindoleacetic Acid</td>
<td>8</td>
<td>83</td>
</tr>
<tr>
<td>labtab</td>
<td>3</td>
<td>5-Hydroxytryptamine</td>
<td>2</td>
<td>55</td>
</tr>
<tr>
<td>labtab</td>
<td>4</td>
<td>17-Hydroxyprogesterone</td>
<td>37</td>
<td>84</td>
</tr>
<tr>
<td>labtab</td>
<td>5</td>
<td>18-Hydroxycorticosterone</td>
<td>20</td>
<td>86</td>
</tr>
<tr>
<td>labtab</td>
<td>6</td>
<td>25-Hydroxy Vitamin D</td>
<td>38</td>
<td>52</td>
</tr>
<tr>
<td>labtab</td>
<td>7</td>
<td>Haemophilus ducreyi</td>
<td>32</td>
<td>62</td>
</tr>
<tr>
<td>labtab</td>
<td>8</td>
<td>Haldol</td>
<td>31</td>
<td>62</td>
</tr>
<tr>
<td>labtab</td>
<td>9</td>
<td>Haloperidol</td>
<td>12</td>
<td>78</td>
</tr>
</tbody>
</table>
```

### labs
```
<table>
<thead>
<tr>
<th>gbl</th>
<th>a1</th>
<th>a2</th>
<th>a3</th>
<th>a4</th>
<th>a5</th>
</tr>
</thead>
<tbody>
<tr>
<td>labs</td>
<td>101000</td>
<td>411</td>
<td>1978-06-06</td>
<td>03:30:00</td>
<td>58</td>
</tr>
<tr>
<td>labs</td>
<td>102000</td>
<td>394</td>
<td>1967-07-21</td>
<td>02:51:00</td>
<td>28</td>
</tr>
<tr>
<td>labs</td>
<td>103000</td>
<td>374</td>
<td>1989-01-07</td>
<td>04:03:00</td>
<td>20</td>
</tr>
<tr>
<td>labs</td>
<td>104000</td>
<td>149</td>
<td>1982-11-10</td>
<td>19:22:00</td>
<td>1</td>
</tr>
<tr>
<td>labs</td>
<td>105000</td>
<td>180</td>
<td>1955-06-21</td>
<td>22:44:00</td>
<td>96</td>
</tr>
<tr>
<td>labs</td>
<td>106000</td>
<td>39</td>
<td>1994-02-08</td>
<td>08:24:00</td>
<td>91</td>
</tr>
<tr>
<td>labs</td>
<td>107000</td>
<td>320</td>
<td>1960-09-13</td>
<td>01:38:00</td>
<td>89</td>
</tr>
<tr>
<td>labs</td>
<td>108000</td>
<td>298</td>
<td>1968-02-19</td>
<td>08:17:00</td>
<td>4</td>
</tr>
<tr>
<td>labs</td>
<td>109000</td>
<td>226</td>
<td>1953-03-17</td>
<td>11:57:00</td>
<td>31</td>
</tr>
</tbody>
</table>
```

### probtab
```
<table>
<thead>
<tr>
<th>gbl</th>
<th>a1</th>
<th>a2</th>
</tr>
</thead>
<tbody>
<tr>
<td>probtab</td>
<td>icd_code</td>
<td>diagnosis</td>
</tr>
<tr>
<td>probtab</td>
<td>001</td>
<td>Cholera</td>
</tr>
<tr>
<td>probtab</td>
<td>002</td>
<td>Typhoid</td>
</tr>
<tr>
<td>probtab</td>
<td>003</td>
<td>Other</td>
</tr>
<tr>
<td>probtab</td>
<td>004</td>
<td>Shigellosis</td>
</tr>
<tr>
<td>probtab</td>
<td>005</td>
<td>Other</td>
</tr>
<tr>
<td>probtab</td>
<td>006</td>
<td>Amebiasis</td>
</tr>
<tr>
<td>probtab</td>
<td>007</td>
<td>Other</td>
</tr>
<tr>
<td>probtab</td>
<td>008</td>
<td>Intestinal</td>
</tr>
<tr>
<td>probtab</td>
<td>009</td>
<td>Ill-defined</td>
</tr>
</tbody>
</table>
```

### problems

36
| problems | 101000 | 117 | 1960-03-16 | 1989-02-20 |
| problems | 102000 | 494 | 1955-06-06 | 1984-09-07 |
| problems | 103000 | 065 | 1971-11-18 | 1995-01-20 |
| problems | 105000 | 084 | 1966-11-02 | 1995-11-29 |
| problems | 106000 | 504 | 1972-02-15 | 1973-09-29 |
| problems | 107000 | 490 | 1963-01-06 | 1957-02-28 |
| problems | 108000 | 086 | 1982-03-08 | 1988-11-12 |
| problems | 109000 | 132 | 1957-07-06 | 1993-09-29 |

| medtab
| gbl | a1 | a2 | a3 | a4 |
| medtab | 10000 | 16 | 44 | ACEPROMAZINE |
| medtab | 10001 | 18 | 66 | ACTIGALL |
| medtab | 10002 | 10 | 77 | ADEQUAN |
| medtab | 10003 | 14 | 92 | ADVANTAGE |
| medtab | 10004 | 084 | 1966-11-02 | 1995-11-29 |
| medtab | 10005 | 18 | 85 | ALBON |
| medtab | 10006 | 4 | 63 | ALLOPURINOL |
| medtab | 10007 | 8 | 90 | AMITRIPTYLINE |
| medtab | 10008 | 4 | 99 | AMLODIPINE |

| medadmin
| gbl | a1 | a2 | a3 | a4 | a5 |
| medadmin | 101000 | 209 | 186 | 1951-06-19 | 1951-06-19 |
| medadmin | 102000 | 12 | 5 | 1956-11-19 | 1956-11-19 |
| medadmin | 103000 | 118 | 36 | 1985-07-16 | 1985-07-16 |
| medadmin | 104000 | 151 | 132 | 1988-11-17 | 1988-11-17 |
| medadmin | 105000 | 55 | 188 | 1990-01-26 | 1990-01-26 |
| medadmin | 106000 | 160 | 50 | 1983-02-09 | 1983-02-09 |
| medadmin | 108000 | 120 | 199 | 1978-09-26 | 1978-09-26 |
| medadmin | 109000 | 57 | 80 | 1984-06-30 | 1984-06-30 |

| patientid
| gbl | a1 | a2 | a3 | a4 | a5 | a6 | a7 | a8 | a9 |
| patientid | 101000 | 1951-02-14 | 15:27:00 | 956ParkLane | Russell,Manitoba | 36564 | (165)564-3155 | 180 |
| patientid | 102000 | 1961-06-07 | 21:24:00 | 969PlaceAve.Apt56 | Stratford,Quebec | 40763 | (971)672-2871 | 999 |
| patientid | 103000 | 1964-02-07 | 19:06:00 | 44DriveLane | IdahoSprings,Colorado | 17138 | (118)638-3065 | 130 |
| patientid | 104000 | 1966-10-16 | 07:51:00 | 9RoundLaneApt86 | VirginiaCity,Nevada | 17934 | (173)673-2722 | 100 |
| patientid | 105000 | 1990-04-12 | 04:03:00 | 928DriveAve. | TennesseeCity,Tennessee | 43991 | (640)580-5543 | 100 |
| patientid | 106000 | 1968-11-17 | 22:39:00 | 728LandingRd. | WashingtonPark,Florida | 35705 | (121)817-3142 | 120 |
| patientid | 107000 | 1966-09-10 | 06:49:00 | 320EstatesRd.Apt68 | Bath,NewBrunswick | 96850 | (960)218-2851 | 100 |
| patientid | 108000 | 1991-08-04 | 15:28:00 | 948LakeRd.Apt66 | California,Kentucky | 45850 | (405)952-5216 | 140 |
| patientid | 109000 | 1996-04-23 | 05:19:00 | 21CrestRd. | Russell,Manitoba | 2139 | (876)963-5443 | 150 |

| ptname
| gbl | a1 | a2 |
| ptname | 101000 | Cynthia Adcock |
| ptname | 102000 | Cherise Alington |
| ptname | 103000 | Fortune Baxter |
| ptname | 104000 | Jinny Ashbaugh |
| ptname | 105000 | Iantha Casteel |
| ptname | 106000 | Isolda Candles |
| ptname | 107000 | Hollie Burney |
| ptname | 108000 | Serrena Hunter |
| ptname | 109000 | Brooklyn Close |
Note that the first column (\textit{gbl}) contains the name of the global array and that the last column contains data stored at the array, if any. The column is empty (null) if no data is stored.

Suppose you wanted to write a Mumps program to display for each patient, the patient's name and any lab tests and results. This can be done as:

```mumps
#!/usr/bin/mumps
set $ztable="labs"
for ptid=$order(^labs(ptid)) do
  . for test=$order(^labs(ptid,test)) do
    .. write "ptid=",ptid," name=",^ptname(ptid)," lab test code=",test," lab test name=",^labtab(test,")),!
  set $ztable="labs"
```

which displays:

```
ptid=1000000 name=Kemp Widaman lab test code=151 lab test name=TofranilWater Deprivation Test
ptid=1001000 name=Marylyn Hindman lab test code=355 lab test name=Chlamydia Test
ptid=1002000 name=Gussie Dickerson lab test code=266 lab test name=Alcohol Screen - Blood
ptid=1003000 name=Lenox Mcmichaels lab test code=1 lab test name=5-HIAA
ptid=1004000 name=Laureen Tanner lab test code=27 lab test name=Hemoglobin
ptid=1005000 name=Hamilton Reade lab test code=91 lab test name=Molecular Diagnostic Tests
ptid=1006000 name=Netta Garratt lab test code=218 lab test name=Rickettsiae Serology
ptid=1008000 name=Rebekah Wynter lab test code=349 lab test name=Ceruloplasmin
ptid=1009000 name=Kemp Widaman lab test code=176 lab test name=Enteric Pathogen Stool Culture
ptid=1010000 name=Cynthia Adcock lab test code=411 lab test name=VIP
ptid=1010000 name=Mydia lab test code=389 lab test name=Osmolality - Urine
```

Notice the use of \textit{$ztable} to switch from one RDBMS table to another. Note: this is not necessary if all the globals are stored in a single table. However, by storing the globals in separate table, it is more easy to query them with ordinary SQL commands. While the SQL commands can use the Mumps column headings (\textit{gbl, a1, a2, a3...}) it is probably easier to create permanent or temporary \textit{VIEWS} with more familiar columns headings (Note: MySQL at this writing only has permanent \textit{VIEWS}).
views). These views can then be used to manipulate the tables. Note: in the following the keyword temporary is not accepted by MySQL and a series of DROP VIEW IF EXISTS lines are needed to remove any earlier instances.

```sql
create temporary view lab_table as select a1 as test_code, a2 as test_name, a3 as low, a4 as high from labtab;
create temporary view lab_rslts as select a1 as ptid, a2 as test, a3 as date, a4 as time, a5 as rslt from labs;
create temporary view problem_table as select a1 as icd, a2 as dx from probtab;
create temporary view problem_list as select a1 as ptid, a2 as icd, a3 as onset, a4 as resolved from problems;
create temporary view med_table as select a1 as medcode, a2 as lowdose, a3 as highdose, a4 as med_name from medtab;
create temporary view med_admin as select a1 as ptid, a2 as medcode, a3 as dose, a4 as date, a5 as time from medadmin;
create temporary view patient_id as select a1 as ptid, a2 as date, a3 as time, a4 as street, a5 as citystate, a6 as zip, a7 as tel, a8 as insurer from patientid;
create temporary view pt_name as select a1 as ptid, a2 as name from ptname;
create temporary view pt_namex as select a1 as ptid, a2 as name, test 'empty' as a3 from ptname;
create temporary view vital_signs as select a1 as ptid, a2 as date, a3 as time, a4 as sys, a5 as dia, a6 as resp, a7 as temp from vitals;
select * from lab_table limit 10;
select * from lab_rslts limit 10;
select * from problem_table limit 10;
select * from problem_list limit 10;
select * from med_table limit 10;
select * from med_admin limit 10;
select * from patient_id limit 10;
select * from pt_name limit 10;
select * from vital_signs limit 10;
select patient_id.ptid, pt_name.name, lab_table.test_name
from pt_name,patient_id,lab_table,lab_rslts
where patient_id.ptid = lab_rslts.ptid
and pt_name.ptid=patient_id.ptid
and lab_rslts.test = lab_table.test_code;
```

SQL Commands to display for each patient, the patient's name and any lab tests and results which results in (base table output omitted). Results are not ordered the same as in the Mumps case.

<table>
<thead>
<tr>
<th>Patient ID</th>
<th>Name</th>
<th>Test Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>101000</td>
<td>Cynthia Adcock</td>
<td>VIP</td>
</tr>
<tr>
<td>102000</td>
<td>Cherise Alington</td>
<td>Oxalate</td>
</tr>
<tr>
<td>103000</td>
<td>Fortune Baxter</td>
<td>CK</td>
</tr>
<tr>
<td>104000</td>
<td>Jinny Ashbaugh</td>
<td>Tobramycin - CSF</td>
</tr>
<tr>
<td>105000</td>
<td>Iantha Casteel</td>
<td>Eosinophil Count - Urine</td>
</tr>
<tr>
<td>106000</td>
<td>Isolda Candles</td>
<td>Hemostasis Study</td>
</tr>
<tr>
<td>107000</td>
<td>Hollie Burney</td>
<td>Candida Vincents Organisms</td>
</tr>
<tr>
<td>108000</td>
<td>Serrena Hunter</td>
<td>Ammonia - Urine</td>
</tr>
<tr>
<td>109000</td>
<td>Brooklyn Close</td>
<td>Ketone Screen</td>
</tr>
<tr>
<td>110000</td>
<td>Serena Wells</td>
<td>Mitochondrial Antibodies</td>
</tr>
<tr>
<td>111000</td>
<td>Chantel Elliott</td>
<td>Aminolevulinic Acid</td>
</tr>
<tr>
<td>112000</td>
<td>Abigayle Moonshower</td>
<td>Allergy Testing, IgE</td>
</tr>
<tr>
<td>113000</td>
<td>Lela Mcmullen</td>
<td>Alpha-1 Microglobulin</td>
</tr>
<tr>
<td>114000</td>
<td>Molly Vorrasi</td>
<td>Bordetella pertussis by PCR</td>
</tr>
<tr>
<td>115000</td>
<td>Humbert Butt</td>
<td>Red Cell Count (RBC) / Indicies (MCHC, MCV, RDW)</td>
</tr>
<tr>
<td>116000</td>
<td>Ceara Davis</td>
<td>Alcohol - Blood</td>
</tr>
</tbody>
</table>
### 3.8.2 Viewing RDBMS Database Tables in Mumps

An ordinary RDBMS table can be viewed as a Mumps global array if a VIEW of the table is created that is compatible with Mumps RDBMS global array access. For example, the following creates a small but typical RDBMS table and then maps a view to it that renames the columns to be compatible with Mumps. Note that the first column of the view, column *gbl*, contains the constant string *mtemps* in each tuple of the view. This will be the global array name. Also, the final column, *a4*, which is where Mumps normally stores any value stored at a global array reference, is a constant empty string in each tuple.

```sql
drop table if exists temps;
cREATE TABLE temps ( city varchar(32), temp int, dewpt int);
insert into temps values ('Boston', '32', '25');
insert into temps values ('Hyannis', '42', '32');
insert into temps values ('Norwood', '32', '12');
insert into temps values ('Quincy', '32', '24');
insert into temps values ('Waltham', '28', '23');
drop view if exists mtemps;
create view mtemps (gbl,a1,a2,a3,a4) as
select 'mtemps' as gbl, city as a1, temp as a2,
dewpt as a3, '' as a4 from temps;

select * from mtemps;
```

A mumps program could access the above with:

```mumps
#!/usr/bin/mumps
set $ztable="mtemps"
write "$ztable",!
for city=$order(temps(city)) do
 . for temp=$order(temps(city,temp)) do
 . . for dewpt=$order(temps(city,temp,dewpt)) do
 ... write city," ",temp," ",dewpt,!
```

which produces:

```
temps
Boston 32 25
Hyannis 42 32
Norwood 32 12
Quincy 32 24
Waltham 28 23
```

Using a similar technique, most RDBMS tables can be viewed in Mumps. Note: MySQL and PostgreSQL at this time do not permit alteration of view values so the tables are read-only.

### 3.8.3 Running SQL Commands in Mumps

You may execute SQL commands from Mumps and retrieve the results. For example:

```mumps
#!/usr/bin/mumps
shell/p mysql -u mumps -D mumps --execute="select * from problems limit 10;"
for do
 . use 6
 . read a
 . if '$test use 5 break
 . use 5
 . write a,!
```
In the above, a command is sent to the MySQL CLI interpreter (a similar one is available for PostgreSQL) and the results are piped back and read by the Mumps program producing the following <tab> separated output (other separators can be requested):

```
bash-4.2# shell.mps
<table>
<thead>
<tr>
<th>gbl</th>
<th>a1</th>
<th>a2</th>
<th>a3</th>
<th>a4</th>
</tr>
</thead>
<tbody>
<tr>
<td>problems</td>
<td>ptid</td>
<td>icd</td>
<td>onset</td>
<td>resolved</td>
</tr>
<tr>
<td>problems</td>
<td>101000</td>
<td>117</td>
<td>1960-03-16</td>
<td>1989-02-20</td>
</tr>
<tr>
<td>problems</td>
<td>102000</td>
<td>494</td>
<td>1955-06-06</td>
<td>1984-09-07</td>
</tr>
<tr>
<td>problems</td>
<td>103000</td>
<td>065</td>
<td>1971-11-18</td>
<td>1995-01-20</td>
</tr>
<tr>
<td>problems</td>
<td>105000</td>
<td>084</td>
<td>1966-11-02</td>
<td>1995-11-29</td>
</tr>
<tr>
<td>problems</td>
<td>106000</td>
<td>504</td>
<td>1972-02-15</td>
<td>1973-09-29</td>
</tr>
<tr>
<td>problems</td>
<td>107000</td>
<td>490</td>
<td>1963-01-06</td>
<td>1957-02-28</td>
</tr>
<tr>
<td>problems</td>
<td>108000</td>
<td>086</td>
<td>1982-03-08</td>
<td>1988-11-12</td>
</tr>
<tr>
<td>problems</td>
<td>109000</td>
<td>132</td>
<td>1957-07-06</td>
<td>1993-09-29</td>
</tr>
</tbody>
</table>
```

### 3.8.4 Access via a Web Server

```bash
#!/usr/bin/mumps

html Content-type: text/html &!&!
html <html><body bgcolor=silver><font size=+1>
  set x="host=test.cs.uni.edu dbname=medical"
  # Open the connection.
  # The &~exp~ causes the result of 'exp' to be inserted into the line
  sql/d &~x~
  # $test will be 1 and $zsql will be 'ok' if it worked
  if $test html Connection to database open <br>
  else do
    . html Connection to database failed </body></html>
    . halt
  # Flush/delete/create a mumps table in the database.
  sql/f
  if $test html Mumps tables initialized <br>
  else html Mumps table initialization failed <br>
  # prepare a query and run it. Output will go to xxx.file
  # where 'xxx' is the process id of this program
  set x="select * from ptname;"
  html Sending query: &~x~ <br>
  sql/o="/tmp/"$_$job_".file" &~x~
  set ptname=$zsqlCols // gets column names TAB separated
  if $test html Query successfully processed <p>
  else do
    . html Query failed. Message=&~$zsql~ <br></body></html>
    . halt
  # Open the file or results and process same.
```
open 1:="/tmp/"$_job_".file,old"

if 'test do
   html Results file error<br></body></html>
   halt
# Each line consists of columns separated by TAB characters
#$char(9) is a TAB. separate the lines. ptname is:

html <table border><tr>
   for i=1:1:6 do
      html <td> &~$piece(ptname,$char(9),i)~</td>
   html </tr>
for do
   . use 1      // unit 1 to be used for I/O
   . read line
   . if 'test break
   . use 5      // unit 5 now used for I/O
   . html <tr>
   . for i=1:1 do
      .. set col=$piece(line,$char(9),i)
      .. if col="" break
      . html <td> &~col~</td>
   . html </tr>
   . html </table>
use 5
html </table>

html </body>
html </html>

shell/g rm &="/tmp/"$_job_".file"~
halt

Gives the following web browser display:
<table>
<thead>
<tr>
<th>id</th>
<th>namefirst</th>
<th>namelast</th>
<th>namemiddle</th>
<th>nameprefix</th>
<th>namesuffix</th>
</tr>
</thead>
<tbody>
<tr>
<td>1002</td>
<td>Sara</td>
<td>Smith</td>
<td>Mary</td>
<td>Mr.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>date</th>
<th>time</th>
<th>test</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1002</td>
<td>1999-Jul-11</td>
<td>12:19</td>
<td>Glucose</td>
<td>10</td>
</tr>
<tr>
<td>1002</td>
<td>1999-Jul-12</td>
<td>12:21</td>
<td>Glucose</td>
<td>12</td>
</tr>
<tr>
<td>1002</td>
<td>1999-Jul-13</td>
<td>12:22</td>
<td>Glucose</td>
<td>14</td>
</tr>
<tr>
<td>1002</td>
<td>1999-Jul-14</td>
<td>12:23</td>
<td>Glucose</td>
<td>16</td>
</tr>
<tr>
<td>1002</td>
<td>1999-Jul-15</td>
<td>12:24</td>
<td>Glucose</td>
<td>18</td>
</tr>
<tr>
<td>1002</td>
<td>1999-Jul-16</td>
<td>12:25</td>
<td>Glucose</td>
<td>20</td>
</tr>
<tr>
<td>1002</td>
<td>1999-Jul-17</td>
<td>12:26</td>
<td>Glucose</td>
<td>22</td>
</tr>
<tr>
<td>1002</td>
<td>1999-Jul-18</td>
<td>12:27</td>
<td>Glucose</td>
<td>24</td>
</tr>
<tr>
<td>1002</td>
<td>1999-Jul-19</td>
<td>12:28</td>
<td>Glucose</td>
<td>16</td>
</tr>
<tr>
<td>1002</td>
<td>1999-Jul-11</td>
<td>12:19</td>
<td>Hct</td>
<td>10</td>
</tr>
<tr>
<td>1002</td>
<td>1999-Jul-12</td>
<td>12:21</td>
<td>Hct</td>
<td>12</td>
</tr>
<tr>
<td>1002</td>
<td>1999-Jul-13</td>
<td>12:22</td>
<td>Hct</td>
<td>14</td>
</tr>
<tr>
<td>1002</td>
<td>1999-Jul-14</td>
<td>12:23</td>
<td>Hct</td>
<td>16</td>
</tr>
<tr>
<td>1002</td>
<td>1999-Jul-15</td>
<td>12:24</td>
<td>Hct</td>
<td>18</td>
</tr>
<tr>
<td>1002</td>
<td>1999-Jul-17</td>
<td>12:26</td>
<td>Hct</td>
<td>22</td>
</tr>
<tr>
<td>1002</td>
<td>1999-Jul-18</td>
<td>12:27</td>
<td>Hct</td>
<td>24</td>
</tr>
<tr>
<td>1002</td>
<td>1999-Jul-19</td>
<td>12:28</td>
<td>Hct</td>
<td>16</td>
</tr>
<tr>
<td>1002</td>
<td>1999-Jul-11</td>
<td>12:19</td>
<td>Hgb</td>
<td>10</td>
</tr>
<tr>
<td>1002</td>
<td>1999-Jul-12</td>
<td>12:21</td>
<td>Hgb</td>
<td>12</td>
</tr>
<tr>
<td>1002</td>
<td>1999-Jul-13</td>
<td>12:22</td>
<td>Hgb</td>
<td>14</td>
</tr>
<tr>
<td>1002</td>
<td>1999-Jul-14</td>
<td>12:23</td>
<td>Hgb</td>
<td>16</td>
</tr>
<tr>
<td>1002</td>
<td>1999-Jul-15</td>
<td>12:24</td>
<td>Hgb</td>
<td>18</td>
</tr>
<tr>
<td>1002</td>
<td>1999-Jul-16</td>
<td>12:25</td>
<td>Hgb</td>
<td>20</td>
</tr>
<tr>
<td>1002</td>
<td>1999-Jul-17</td>
<td>12:26</td>
<td>Hgb</td>
<td>22</td>
</tr>
<tr>
<td>1002</td>
<td>1999-Jul-18</td>
<td>12:27</td>
<td>Hgb</td>
<td>24</td>
</tr>
<tr>
<td>1002</td>
<td>1999-Jul-19</td>
<td>12:28</td>
<td>Hgb</td>
<td>16</td>
</tr>
</tbody>
</table>
4 Implementation Notes

4.1 Blanks in numeric strings
   Leading blanks in numeric strings are ignored.

4.2 PostgreSQL Userid
   The userid of the user running a Mumps program that accesses PostgreSQL must be an
   authorized user of PostgreSQL. See createuser.

4.3 PostgreSQL Transaction Limit
   The number of pending transactions between SQL Begin and SQL Commit commands is limited by
   PostgreSQL. If this number (implementation defined) is exceeded, transactions will be lost. Check
   the PostgreSQL log in /var/log/postgresql.

4.4 Maximum number of global indices
   No more than 10 indices are permitted in a global array reference when using PostgreSQL as the
   backend global array data store.

4.5 Lock command with PostgreSQL
   Locks are not needed if using the PostgreSQL global array store as SQL transaction commands can
   achieve the same effect.

4.6 Naked indicator
   This version of Mumps does not support the naked indicator. The naked indicator has no place in a
   modern or even semi-modern programming language. It was originally included in early versions of
   Mumps because of the inefficient binary mapping of an n-way tree which was used at the time to
   store the global arrays. The naked indicator was a short-hand to the interpreter to allow it to search
   for a global without stating at the top of the tree each time thus resulting in faster access. That is no
   longer the case with modern B-tree based access methods. Another issue is the perceived ambiguity
   of determining what exactly the naked indicator is after certain Mumps operations. Unfortunately,
   some legacy applications use it. These should be re-written.

4.7 Lock command in client/server mode
   In single user mode (native global database, not shared), the Lock command has no meaning. In
   native client server mode, the Lock command creates a file named Mumps.Locks in /tmp where the
   lock information for the system is stored. If this file becomes corrupted due to abnormal
   terminations, it should be deleted. It will be rebuilt as needed.
   When using PostgreSQL for the backend global array store, the Lock command works the same as it
   does in native client server mode but the user would be strongly advised to use the more modern
   native SQL transaction processing commands (BEGIN, COMMIT, ROLLBACK, etc.) to achieve the
   same effect with far greater integrity.

4.8 Job command
   The JOB command results in a C/C++ fork() function to be executed thus creating a child process.
   The child process will attempt to execute the argument to the JOB command. The JOB command
   may be used in single user mode but only one process may access the globals if the single user
   native global array option is in effect. In client server modes, this restriction is not in effect.
   The child process must end with a HALT command or the child process will hang.
4.9 Client server common errors

A common source of problems in native client server mode is failing to start the server or attempting to access the server from a non-authorized userid.

4.10 File names containing directory information

When invoking a file name containing directory information (forward slash in Linux and backslash in DOS) with the DO or GOTO commands, the file name must be enclosed in quotes. For example:

```mumps
set x=""^/home/user/xxx.mps"" goto @y
go to ""^/home/user/xxx.mps"
```

Note the extra quotes. These are required.

4.11 File Names

File names should conform to variable naming conventions except that the first character of a file name may not be the per cent sign (%). The first character must be alphabetic. File names may only contain letters, digits and the per cent sign.

4.12 Array Index Collating Sequence

Array index collating sequences for both global and local array is ASCII. That is, for the $query() and $order() functions, all array indices will be presented in the same order as ASCII strings. Thus, in an array with 15 elements whose indices range from 1 to 15, the indices will be presented as:

```
1 10 11 12 13 14 15 2 3 4 5 6 7 8 9
```

Other versions of Mumps may present numeric indices in numeric order. This, however, leads to considerable inefficiencies in the data base.

You may achieve numeric ordering by storing the indices padded to left with blanks such as:

```mumps
for i=1:1:15 set ^a($justify(i,8))=i
set i="" for set i=$order(^a(i)) quit:i='' write +i," "
```

the indices will now be presented as:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Note the the +i in the write command has the effect of converting the string to a number with no leading blanks.

4.13 Subroutine & Function Calls

Subroutines and functions may be performed in several ways as shown in Figure 1. Values returned from functions invoked by a do command are ignored. In standard Mumps, the $$ form is used only with function invocations.

Caution: be certain to include a halt or other exit in your program prior to any functions. If the halt is not present, function code will be entered and any passed variables will be undefined.

```mumps
#!/usr/bin/mumps
# calls.mps

set i=10
do fcn(i)
do fcn(5)
do $$fcn(i)
do $$fcn(5)
set k=$$fcn(5)
write "returned k=",k,!
set i=10
do fcn^ext.mps(i)
```
do fcn"ext.mps(5)
do $$fcn"ext.mps(i)
do $$fcn"ext.mps(5)
set k= $$fcn"ext.mps(5)
write "returned k=",k,!

do fcn"ext1.mps
do fcn"ext1.mps
do $$fcn"ext1.mps
do $$fcn"ext1.mps
set k= $$fcn"ext1.mps
write "returned k=",k,!

halt

cxn(x) write "in fcn(x) value passed is ",x,!
quit x

----------------------------------------

#!/usr/bin/mumps
# ext.mps

cxn(x) write "in fcn(x) value passed is ",x,!
quit x

----------------------------------------

#!/usr/bin/mumps
# ext1.mps

cxn write "in fcn ext1.mps",!
set x=22
quit x

----------------------------------------

output results:
in fcn(x) value passed is 10
in fcn(x) value passed is 5
in fcn(x) value passed is 10
in fcn(x) value passed is 5
in fcn(x) value passed is 5
returned k=5
in fcn(x) value passed is 10
in fcn(x) value passed is 5
in fcn(x) value passed is 10
in fcn(x) value passed is 5
in fcn(x) value passed is 5
returned k=5
in fcn ext1.mps
in fcn ext1.mps
in fcn ext1.mps
in fcn ext1.mps
in fcn ext1.mps
returned k=22

Figure 1 Subroutine/Function Calls

4.14 $Fnumber() Function

The $Fnumber() is implemented via the C function strfmon() which provides much greater flexibility when dealing with differing locales and, especially, currencies. The default locale is en_US.UTF-8 but this can be set with the configure option:

--with-locale=location-information

You may use $Fnumber() with the legacy Mumps parameters or use it with a pattern parameter designed for strfmon().

If you use the strfmon() parameter option, the function takes two arguments. The first must be a number consisting of only numeric characters. The second is a character string conforming to a
strfmon() pattern but preceded by an asterisk to distinguish the pattern from those used by the legacy Mumps function of the same name. The strfmon() function is well documented but here are some examples:

```
set x=12345.6789
write $fn(x,"*%!n")  ==>  12,345.68
write $fn(x,"*%n")    ==>  $12,345.68
write $fn(x,"*%i")    ==>  USD 12,345.68
write $fn(x,"*%n3")   ==>  $12,345.683
write $fn(x,"*%20n")  ==>  $12,345.68
```

4.15 $select() Function

All arguments of the $select() function are evaluated.

4.16 Compiling Large Programs

When compiling large programs, especially if MySQL is enabled, there may be a warning about variable tracking from the gcc/g++ compiler. You may ignore this.

4.17 Embedded Expressions

In several extended Mumps commands the figure &~exp~ may appear. The expression is evaluated and the result replaces the figure. For example:

```
set x="ls -lh"
shell &~x~

set x= "select * from abc;"
sql &~x~
```

2 Using the compiler is not presently recommended.
5 Shell Command

5.1 shell

5.2 shell/g

5.3 shell/p

The shell command passes the remainder of the line to a shell for execution (sh in Linux). Shell output will appear on stdout. The command sets $test to false if the fork() fails, true otherwise.

This command is not presently available in the DOS version.

The shell/p form passes the remainder of the line to a shell for execution but opens a pipe from the shell to Mumps unit number 6. All stdout output from the shell is directed to unit number 6 and can be read with any of the input commands or functions in association with the use command.

The shell/g form passes the remainder of the line to a shell for execution (sh in Linux) and opens a pipe from the Mumps program to the shell as Mumps unit number 6. Data written to this unit becomes stdin to the shell. Output from the shell is written to stdout. Remember to close unit number 6 to signal end-of-file to the shell.

With no qualifier, the shell command passes the remainder of the command line to a shell. Input or output from the shell come from or go to stdin or stdout, respectively.

In all cases, the remainder of the command line is scanned for &~...~ expressions. The expression between &~ and ~ is evaluated and the result replaces the &~...~ expression.

For example:

```
shell sort dictionary.tmp | uniq -c | sort -nr > dictionary.s
```

The Linux shell created will do the following:

1. The file dictionary.tmp, a collection of words, will be sorted by sort and the output piped to uniq
2. uniq counts duplicate entries and pipes its output consisting of a count and a word to sort
3. sort sorts the result numerically by number of duplicates in reverse order and writes its output to dictionary.s.

```
1  shell/p sort dictionary.tmp | uniq -c | sort -nr
2  open 1:"dictionary.s,new"
3  for  do
4    .  read line
5    .  if  '$test break
6    .  use 1
7    .  write line,!
8  close 1
```

The above does the same but the output will be presented to Mumps unit 6 which reads and writes the result to the file named dictionary.s
6 Added Commands

6.1 Database expr

The database command may be used to set the name of the files to be used to store the native global arrays. The expression will be evaluated and the resulting name will become the name, suffixed .key and .dat, of the files in which the native global arrays are stored. The expression may contain directory information. For example:

database "/home/user/data/mumps"

will cause the system to access files:

/home/user/data/mumps.key
/home/user/data/mumps.dat

for the global array tree and data files. If directory information is omitted, the files will be in the current directory.

This command must be issued prior to any attempt to access the global arrays.

6.2 ZHALT return_code

The zhalt command will terminate the current program with a return error code given by its argument. Example:

if a=0 zhalt 99

The value of $? in the BASH environment will be 99.
7 Z Functions and System Variables

$zfunctions$ are extensions added by the implementor and not covered by the standard. Thus, many if not all of the following M2 extensions may not be supported or supported differently in other implementations. Likewise, there are implementer defined system variables which may be queried and, in some cases, set.

M2 implementation note: you may add new $z functions by modifying the function $zfcn() located in the source file $bifs.cpp.in$

7.1 System Variables

7.1.1 $zttable$

The value in $zttable$ is the name of the current database table if you are using PostgreSQL or MySQL to store the global arrays. It has no meaning if you are using the native Btree. The default value will be $mumps$ unless this was changed during the $configure$ step or changed during execution.

You may set the name of the database table in use by setting $zttable$. All database references will take place in the table you set $zttable$ to until you change it again or the program terminates.

When you start a Mumps program, $zttable$ reverts to the value set by $configure$.

The string returned by $zttable$ consists of two parts separated by a comma. The first part is the name of the table and the second part is the number of columns available for global array indices. The number is indices is two less than the actual number of columns as one column is reserved for the global array name named $gbl$, and one column for the value stored at the global reference named $ax$ where the value of $x$ is one greater that the number of indices.

If you change the default table, you need to insure that it exists and is properly defined for use my Mumps. The $sqlf$ command will create and initialize a new table or re-initialize to empty an existing table. Warning messages may appear the first time you create a table.

7.1.2 $zttabsize$

The maximum number of indices permitted in a global array reference. When queried, this variable returns the current setting. It may be set (maximum of 20). If you set it, you must initialize the global array before using it or errors will result.

7.2 Math Functions

The following C/C++ math functions are available in M2. Their arguments and return values are the same as the correspondingly named C++ functions.

7.2.1 $zabs(arg)$ absolute value

Function returns the absolute value of its numeric argument.

7.2.2 $zacos(arg)$ arc cosine

Computes the inverse cosine (arc cosine) of the input value. Arguments must be in the range -1 to 1.

7.2.3 $zasin(arg)$ arc sine
Computes the inverse sine (arc sine) of the argument $\text{arg}$. Arguments must be in the range -1 to 1.

7.2.4 $\text{atan}(\text{arg})$ arc tangent
Computes the inverse tangent (arc tangent) of the input value.

7.2.5 $\text{zcos}(\text{arg})$ cosine
Computes the cosine of the argument $\text{arg}$. Angles are specified in radians.

7.2.6 $\text{zexp}(\text{arg})$ exponential
Calculates the exponential of $\text{arg}$, that is, $e$ raised to the power $\text{arg}$ (where $e$ is the base of the natural system of logarithms, approximately 2.71828).

7.2.7 $\text{zlog}(\text{arg})$ natural log
Returns the natural logarithm of $\text{arg}$, that is, its logarithm base $e$ (where $e$ is the base of the natural system of logarithms, 2.71828...).

7.2.8 $\text{zlog10}(\text{arg})$ base 10 log
Returns the base 10 logarithm of $\text{arg}$.

7.2.9 $\text{zpow}(\text{arg1},\text{arg2})$ power function
Calculates $\text{arg1}$ raised to the exponent $\text{arg2}$.

7.2.10 $\text{zsqr}(\text{arg})$ square root
Function returns the square root of its numeric argument.

7.2.11 $\text{zsin}(\text{arg})$ sine function
Computes the sine of the argument $\text{arg}$. Angles are specified in radians.

7.2.12 $\text{zsqr}(\text{arg})$ square function
Function returns the square of its numeric argument.

7.2.13 $\text{ztan}(\text{arg})$ tangent function
Computes the tangent of $\text{arg}$.

7.3 Date functions

7.3.1 $\text{zdate}(\text{or } \text{zd})$ formatted date string
Function returns the system date and time in standard system printable format. This includes: day of week, month, day of month, time (hour:minute:second), and year (4 digits).

7.3.2 $\text{zd1}$ numeric internal date
Returns the number of seconds since January 1, 1970 - a standard used in Linux. This number may be used to accurately correlate events.

7.3.3 $\text{zd2}\text{(InternalDate)}$ date conversion
Translates the Linux time from $\text{ZD1}$ into standard system printable format. The argument is a Linux format time value.

7.3.4 $\text{zd3}(\text{Year,Month,Day})$ Julian date
Returns the day of the year (Julian date) for the Gregorian date argument.
7.3.5 $zd4(Year,DayOfYear) Julian to Gregorian

Returns the Gregorian date for the Julian date argument.

7.3.6 $zd5(Year, Month, Day) comma listed date

Returns a string consisting of the year, a comma, the day of year, and the number of days since Sunday (Monday is 1).

7.3.7 $zd6 hour:minute

Returns a string consisting of the hour, a colon, and the minute.

7.3.8 $zd7 hyphenated date

Returns a string consisting of the year, hyphen, month, hyphen, and day of month. If an argument is given in the form of the number of seconds since Jan 1, 1970, the result returned will reflect the argument date.

7.3.9 $zd8 hyphenated date with time

Returns a string consisting of the year, hyphen, month, hyphen, and day of month, comma, and time in HH:MM format. If an argument is given in the form of the number of seconds since Jan 1, 1970, the result returned will reflect the argument date.

7.4 Special Purpose Functions

The following special purpose functions are available:

7.4.1 $zb(arg) remove blanks

Function returns a string in which all leading blanks have been removed and all multiple blanks have been replaced by single blanks. See also $zNoBlanks(). Figure 2 gives examples.

```
#!/usr/bin/mumps
set a="   abc   xyz     123    
write $zb(a),"***",!
output:
abc xyz 123 ***
```

Figure 2 - $Zb() examples

7.4.2 $zchdir(directory_path) change directory

Function changes the current directory to the path specified. If the operation succeeds, a zero is returned. If it fails, -1 is returned.

7.4.3 $zCurrentFile Current Mumps File

Returns the name of the currently executing Mumps program file (if any) or blank.

7.4.4 $zdump([filename]) dump global arrays

Function dumps the globals to a sequential ASCII file in the current directory. If an argument is given, it is taken as the name of the file to which the globals will be written. If the argument is omitted, a file name is constructed from the system date of the form number.dmp where number is the value of the C++ time() function at the time of the dump.

The dump file is a pure ASCII text file. Each entry in the global array is represented by two lines. The first line is the global array reference and the second line is the store value. In the global array reference, parentheses and commas are replaced by the "~" character. Thus, if you wish to use this facility, you may not include the "~" character in a global array index.

The function $zrestore() reloads the global arrays from a dump file (see below).
$zdump and $zrestore do not work when PostgreSQL is used for the global array store.

**7.4.5 $zrestore[(arg)]** restore globals

Function restores the globals from a dump file produced by $zdump. If an argument is given, it is taken as the name of the dump file otherwise, the default name dump is used.

$zdump and $zrestore do not work when PostgreSQL is used for the global array store.

**7.4.6 $zfile(arg)** file exists test

Function returns a zero or one indicating if the file given as the argument exists.

**7.4.7 $zflush** flush Btree buffers

Function flushes all modified native global array handler buffers to disk. The function should only be used with the native globals. After flushing, all updates to the btree file system have been committed. In cases where the internal buffers are very large, this function may take several seconds to execute. The function returns the empty string. Flushing the buffers is a precaution against system failure which would otherwise result in corruption of the global arrays.

**7.4.8 $zgetenv(arg)** get environment variable

Returns the contents of the environment variable specified as arg or the empty string if the variable is not found.

**7.4.9 $zhtml(arg)** encode HTML string

Function encodes its argument in the form necessary to be a cgi-bin parameter. That is, alphabetics remain unchanged, blanks become plus signs and all other characters become hexadecimal values, preceded by a percent sign.

**7.4.10 $zhit** global array cache hit ratio

Function calculates and returns the native global array cache hit ratio. This number ranges between zero and one. A value of one indicates all requests were satisfied from the cache while a value of zero indicates no requests were satisfied from the cache. Calling this function resets the hit ratio to zero. A higher value for the hit ratio indicates better database performance.

**7.4.11 $zlower(string)** convert to lower case

Function returns the input string with alphabetics converted to lower case.

**7.4.12 $znormal(arg1[,arg2])** word normalization

Function converts the word passed as argument 1 to lower case and removes any embedded punctuation. If a second argument is given, the word is truncated to the length specified by this argument. If no second argument is given, words are truncated to 25 characters if their length exceeds 25 characters.

**7.4.13 $zNoBlanks(arg)** remove all blanks

Returns arg with all blanks removed. See also: $zb.

**7.4.14 $zp[ad](arg1, arg2)** left justify with padding

Function left justifies the first argument in a string whose length is given by the second argument, padding to the right with blanks.

**7.4.15 $zseek(arg)**

Function takes one argument (a positive integer) which is a byte offset in the currently active (use) file. The command moves the file pointer to that location in the file. $zseek() may only be used on files opened with old attribute. Figure 3 gives examples.
7.4.16 $zsrand(arg)

Seed the random number generator. The value passed as the argument will seed the internal random number generator. If the random number generator is re-seeded with the same seed, the sequence of random numbers produced by $random will be the same. The value passed must be a positive integer.

7.4.17 $zstem(arg)

Returns an word English word stem of the argument. This function attempts to remove common endings from words and return a root stem.

7.4.18 $zsystem(arg)

Executes "arg" in a system shell. Returns -1 (fork failed) or the return code of the execution of the argument. See also the shell command.

7.4.19 $ztell

Function returns the byte offset in the currently open file. Similar to the C++ ftello function. Note: The offset returned is for the file most recently made the default i/o file by the use command. $ztell may be used on either a file opened as new, old or append. (See example under $zseek above)

7.4.20 $zu(expression)

Function returns 1 if the expression is numeric, 0 otherwise.

7.4.21 $zwi(arg)

Function loads an internal buffer with the string given as the argument. The alphabetic characters of the argument are converted to lower case. The contents of this buffer are returned by the $zwn and $zwp functions. Figure 4 gives examples.
7.4.22 $zwn extract words from buffer

Function returns successive words from the internal buffer delimited by blanks. When no more words remain, it returns an empty string (string of length zero). Returned words are converted to lower case. See $zwi.

7.4.23 $zwp extract words from buffer

Function returns successive words from an internal buffer delimited by blanks and punctuation characters. When no more words remain, it returns an empty string (string of length 0). Returned words are converted to lower case. See $zwi.

7.4.24 $zws(string) initialize internal buffer

Initializes the parse buffer but does not convert "string" to lower case as is the case with $zwi

```
1 #!/usr/bin/mumps
2 set i="now, is the time, for all good"
3 set %=$zwi(i)
4 for w=$zwp write w,!
5 write "-------",!
6 set %=$zwi(i)
7 for w=$zwn write w,!
```

output:
```
now, is the time, for all good -------
now, is the time, for all good
```

Figure 4 - $Zwi() examples

7.4.25 Scan Functions

7.4.25.1.1 $zzScan
7.4.25.1.2 $zzScanAlnum
7.4.25.1.3 $zzInput(var)

The functions return the next word in the current input stream delimited by white space. Words are restricted to a maximum length of 1023. Successive calls return successive words. When there are no more input words, an empty string is returned and $test is set to false.

If only part of a line is scanned as a result of these functions, a subsequent read command will begin at the white space following the last word returned.

If scanning input from stdin (i/o unit 5), you may signal end of file with a control-d on a separate line by itself. This will result terminate the scan and $test will be set to false.

$zzScan returns all words delimited by whitespace with no conversion. Words may contain any printable ASCII character.

$zzScanAlnum processes words before returning them according to the following rules:

- Special characters at the beginning of a word are ignored.
- Words beginning with digits are not returned. If a word begins with one or more special characters followed by a digit, it is not returned.
Words shorter than 3 characters or longer than 25 characters are not returned. Words are converted to all lower case characters. If a word contains embedded special characters, it is treated as a delimiter.

Both functions will advance to additional lines as needed. If a word exceeds 1023 bytes, the results are undefined. See Figure 5 for an example.

```
for the input line:
now -- __ ?? !@#$%^&*()_+= IS 2for the time for
  for set i=$zzScan quit:'$test write i,!
output:
  now
  --
  __
  ??
  !@#$%^&*()_+=
  IS
  2for
  the
  time
for
  for set i=$zzScanAlnum quit:'$test write i,!
output:
  now
  the
  time
for
  for i=$zzScanAlnum do
    . write i,!
output:
  now
  the
  time
```

Figure 5 - Scan functions examples

$zzInput(var) reads an entire input line, converts all characters to lower case, separates the words, removes punctuation (as defined by the C ispunct() function except hyphen), and stores the words into a numerically indexed array whose name is the value of the variable or constant passed as the argument. The function returns the number of elements in the array. A return of zero indicates no input was obtained (end of file). As the array created by the function could be quite large, you should probably kill it when no it is longer needed. The maximum line length permitted is twice the system parameter MAX_STR (9,000 bytes by default).

7.5 Vector and Matrix Functions

7.5.1 $zzAvg(vector)
Computes and returns the average of the numeric values in the vector. For example, see Figure 6.

```
!#/usr/bin/mumps
for i=1:1:10 set ^a(99,i)=i
set i=$zzAvg(^a(99))
write "average=",i,!
```

Figure 6 - $zzAvg() example
The above writes 5.5

7.5.2 $zzCentroid(gblMatrix,gblRef)$

A centroid vector $gblRef$ is calculated for the invoking two dimensional global array $gblMatrix$. The centroid vector is the average value for each for each column of the matrix. Any previous contents of the global array named to receive the centroid vector are lost. The global array $gblMatrix$ must contain at least two dimensions. See Figure 7 for an example. The matrix must be a top level global array.

```mumps
#!/usr/bin/mumps
for i=0:1:10 do
  for j=1:1:10 do
    set ^A(i,j)=5
  set %=$zzCentroid(^A,^B)
for i=1:1:10 write ^B(i),!
```

**Figure 7 - $zzCentroid() example**

7.5.3 $zzCount(gblVector)$

Computes and returns the number of numeric values in the vector and any descendants. For example:

```mumps
#!/usr/bin/mumps
kill ^a
for i=1:1:10 set ^a(99,i)=i
set i=$zzCount(^a(99))
write "count=",i,!
```

**Figure 8 - $zzCount() example**

The above writes 10

7.5.4 $zzMax(gbl)$

Computes and returns the maximum numeric value in the vector and any descendants. See Figure 9 for an example.

```mumps
#!/usr/bin/mumps
for i=1:1:10 set ^a(99,i)=i
set i=$zzMax(^a(99))
write "max=",i,!
```

**Figure 9 - $zzMax() example**

The above writes the largest value stored in the vector.
7.5.5 $zzMin(gbl)$

Returns the minimum numeric value stored in the vector and any descendants. See Figure 10 for an example.

```
#!/usr/bin/mumps
for i=1:1:10 set ^a(99,i)=i*2
set i=$zzMin(^a(99))
write "min=",i,!
```

Figure 10 - $zzMin()$ example

7.5.6 $zzMultiply(gbl1,gbl2,gbl3)$

Multiplies the first and second matrix leaving the result in the third. The ordinary rules of algebra apply. Figure 14 gives an example. The arguments $gbl1$ and $gbl2$ must be top level, two dimensional arrays.

7.5.7 $zzSum(gblVector)$

Computes and returns the sum of the numeric values stored in the vector. For example, see Figure 15.

7.5.8 $zzTranspose(gblMatrix1,gblMatrix2)$

Transposes the first global array matrix leaving the result in the second. For example, see Figure 16. The argument $gblMatrix1$ must be a top level, two dimensional array.

7.6 Text Processing Functions

The following functions are used in connection with experiments in information storage and retrieval.

7.6.1 Similarity Functions

7.6.1.1 $zzCosine(gbl1,gbl2)$

7.6.1.2 $zzSim1(gbl1,gbl2)$

7.6.1.3 $zzDice(gbl1,gbl2)$

7.6.1.4 $zzJaccard(gbl1,gbl2)$

These compute the Cosine, Sim1, Dice and Jaccard similarity coefficients between document vectors given as the first and second arguments. Both arguments are numeric global array vectors. The formulae are given in Figure 11 and an example in code is given in Figure 12. The formulae calculate the similarities between two global array vector $gbl1$ and global array vector $gbl2$. The vectors need not be of equal length. Missing elements are interpreted as zero. The vectors should be top level vectors.
Figure 11 - Similarity formulae

\[
\text{Similarity}_\text{Dice}(i, j) = \frac{2 \sum_{k=1}^{t} \text{Term}_{ik} \cdot \text{Term}_{jk}}{\sum_{k=1}^{t} \text{Term}_{ik} + \sum_{k=1}^{t} \text{Term}_{jk}}
\]

\[
\text{Similarity}_\text{Jaccard}(i, j) = \frac{\sum_{k=1}^{t} \text{Term}_{ik} \cdot \text{Term}_{jk}}{\sum_{k=1}^{t} \text{Term}_{ik} + \sum_{k=1}^{t} \text{Term}_{jk} - \sum_{k=1}^{t} (\text{Term}_{ik} \cdot \text{Term}_{jk})}
\]

\[
\text{Similarity}_\text{Cosine}(i, j) = \frac{\sum_{k=1}^{t} \text{Term}_{ik} \cdot \text{Term}_{jk}}{\sqrt{\sum_{k=1}^{t} \text{Term}_{ik}^2 \cdot \sum_{k=1}^{t} \text{Term}_{jk}^2}}
\]

\[
\text{Similarity}_\text{Sim1}(i, j) = \sum_{k=1}^{t} \text{Term}_{ik} \cdot \text{Term}_{jk}
\]

Figure 12 - Similarity functions

7.6.2 $\text{zzBMGSearch}(\text{arg1}, \text{arg2})$

Boyer-Moore-Gosper Function returns the number of non-overlapping occurrences of $\text{arg1}$ in $\text{arg2}$.

These functions, were obtained from
and were written by Jeffrey Mogul (Stanford University), based on code written by James A. Woods (NASA Ames, an agency of the U.S. Government) and are thus believed to be in the public domain. Figure 13 gives an example.

```mumps
#!/usr/bin/mumps
set key="now"
set str="now is the now of the now in the know"
write $zBMGSearch(key,str),!
output:
```

**Figure 13 - $zzBMGSearch() example**

### 7.6.3 $zPerlMatch(string,pattern)

Applies the Perl pattern to string and returns 1 if the pattern fits and 0 otherwise. The $zPerlMatch function has the side effect of creating variables in the local symbol table to hold backreferences, the equivalent concept of $1, $2, $3, ... in Perl. Up to nine backreferences are currently supported, and can be accessed through the same naming scheme as Perl ($1 through $9). These variables remain defined up to a subsequent call to $zPerlMatch, at which point they are replaced by the backreferences captured from that invocation. Undefined backreferences are cleared between invocations; that is, if a match operation captured five backreferences, then $6 through $9 will contain the empty string. Figure 17 contains examples (long lines wrapped).

```mumps
#/usr/bin/mumps
set ^d("1","1")=2
set ^d("1","2")=3
set ^d("2","1")=1
set ^d("2","2")=-1
set ^d("3","1")=0
set ^d("3","2")=4
set ^e("1","1")=5
set ^e("1","2")=-2
set ^e("1","3")=4
set ^e("1","4")=7
set ^e("2","1")=-6
set ^e("2","2")=1
set ^e("2","3")=-3
set ^e("2","4")=0
set %=$zzMultiply(^d,^e,^f)
for i="":$order(^f(i)):"" do
   for j="":$order(^f(i,j)):"" do
      write i,"",j,"",^f(i,j),!
output:
```

**Figure 14 - $zzMultiply() example**
Figure 15 - $zzSum() example

```mumps
define ^a(i) as.integer
set ^a(99) = i
set i = $zzSum(^a(99))
write "sum=", i,!
```

output:

```
55
```

Figure 16 - $zzTranspose() example

```mumps
define ^d(i,j) as.integer
set ^d("1","1") = 2
set ^d("1","2") = 3
set ^d("2","1") = 4
set ^d("2","2") = 0
set %@zzTranspose(^d)^f
for i="":$order(^f(i)):"" do
  . for j="":$order(^f(i,j)):"" do
    . write i," ",j," ",^f(i,j),!
```

output:

```
1 1 2
1 2 4
2 1 3
2 2 0
```

Figure 17 - $zPerlMatch() example

```mumps
write "Please enter a telephone number:",!
read phonenum
set p = "^(1-)?\(\?\d{3}\)?\-?\d{4}$" 
if $zperlmatch(phonenum,p) do
  . write "+++ This looks like a phone number.",!
  . write "The area code is: ",$2,!
else do
  . write "--- This didn't look like a phone number.",!
```

output:

```
Please enter a telephone number:
(123) 456-7890
+++ This looks like a phone number.
The area code is: (123)

Please enter a telephone number:
(123) 456-7890
+++ This looks like a phone number.
```

Figure 18 - $zReplace() example

7.6.4 $zReplace(string,pattern,replacement)

The regular expression in pattern is evaluated on string and, if there is a match, the matching section is replaced by replacement. Figure 18 contains an example. In the first part, the word 'is' is replaced by 'IS'. In the second part, a match is sought for any content between two sets of matching brackets ([...]). The matched section is in back reference $2. This is then used as a pattern to be replaced.
7.6.5 $\text{zShred(string,length)}$

7.6.6 $\text{zShredQuery(string,length)}$

The $\text{zShred()}$ function segments the input argument \texttt{string} into fragments of \texttt{length} size upon successive calls. The function returns a string of length zero when there are no more fragments of size \texttt{length} remaining (thus, short fragments at the end of a string are not returned).

$\text{zShred}$ copies the input string to an internal buffer upon the first call. Subsequent calls retrieve from this buffer. When the buffer is consumed, the function will copy the contents of the next string submitted to the buffer. Figure 19 contains an example.

```mumps
1 #!/usr/bin/mumps
2 set a="now is the time for all"
3 set a=$zReplace(a,"is","IS")
4 write a,!
5
6 set a="[now is the time]"
7 if $zPerlMatch(a,"(\[[\]])\{.*\}\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()\}()}
#!/usr/bin/mumps
set a="now is the time for all good men to come to "
set a=a_"the aid of the party"
for do quit:j=""
  set j=$zShredQuery(a,5)
  if j="" quit
  write j,!

output:

Figure 20 - $ShredQuery() example

7.6.7 $zzSoundex(s1)
Returns the Soundex code for the argument string as follows:

1. All letters are converted to lower case;
2. Non-alphabetic characters are removed;
3. Adjacent duplicate letters are replaced by a single occurrence;
4. The first letter is retained;
5. The letters b, f, p, and v are replaced by the number 1;
6. The letters c, g, j, k, q, s, x, and z are replaced by the number 2;
7. The letters d and t are replaced by the number 3;
8. The letter l is replaced by the number 4;
9. The letters m and n are replaced by the letter 5;
10. The letter r is replaced by the number 6;
11. The is truncated to four characters.

7.6.8 $zSmithWaterman(s1,s2,algn,mat,gap,noMatch,match)
Computes the Smith Waterman score between two strings. Result returned is the highest alignment score achieved. String lengths are limited by STR_MAX in the interpreter. If you compare very long strings (>100,000 characters), you may exceed stack space. This can be increased under Linux with the command:

ulimit -s unlimited

Figure 21 gives an example.

#!/usr/bin/mumps
set s1="now is the time"
set s2="now i th time"
set i=$zSmithWaterman(s1,s2,1,0,-1,-1,2)
write "score=",i,!

output:

1  now- is the time 16
::: :::: :::::
1  now  i- th time 16
score=23
Figure 21 - $zSmithWaterman()

Parameters:

If $algn$ is zero, no printout of alignments is produced. If $algn$ is not zero, a summary of the alternative alignments will be printed.

If $mat$ is zero, intermediate matrices will not be printed.

The parameters $gap$, $noMatch$ and $match$ are the gap and mismatch penalties (negative integers) and the match reward (a positive integer).

If insufficient memory is available, a segmentation violation will be raised. Try increasing your stack size.

7.6.9 $zzIDF(global,doccount)$

Calculates the Inverse Document Frequency score of words contained in the argument $global$. The parameter $doccount$ is the total number of documents. The index of each element of the $global$ vector is a word and the value stored is the number of times the word occurs in the collection. Figure 22 gives and example. The vector argument $global$ must be a top level array.

```
1 #!/usr/bin/mumps
2 set ^a("now")=-2
3 set ^a("is")=-5
4 set ^a("the")=-6
5 set ^a("time")=-3
6 set j=4
7 set %=$zzIDF(^a,j)
8 for i="":$order(^a(i)):"" write i," ",^a(i),!
output:
 is 0.7
now 2.0
the 0.4
time 1.4
```

Figure 22 - $zzIDF()$ example

7.6.10 Correlation Functions

7.6.10.1 $zzTermCorrelate(global1,global2)$

Calculates the Term-Term co-occurrence matrix for the Document-Term matrix in $global1$. The result is placed in $global2$.

A Term-Term matrix has terms (words) as the indices of its rows and columns. A Term-Term matrix gives, for each position, the degree to which the term corresponding to the row is similar to the term corresponding to the column. The diagonal, which is the degree a term is related to itself, is ignored. Both operands must be top level arrays.

In both the doc-doc and term-term matrices, the upper and lower diagonal matrices are mirror images of one another. Figure 23 gives an example. The order of words in the output will depend upon which data base facility is in use and what it's collating settings are. The Native global array handler collates according to ASCII-7.

```
1 #!/usr/bin/mumps
2 kill ^A,^B
3
4 set ^A("1","computer")=-5
5 set ^A("1","data")=-2
6 set ^A("1","program")=-6
7 set ^A("1","disk")=-3
8 set ^A("1","laptop")=-7
9 set ^A("1","monitor")=-1
10
11 set ^A("2","computer")=-5
12 set ^A("2","printer")=-2
```
set "2","program"=6
set "2","memory"=3
set "2","laptop"=7
set "2","language"=1

set "3","computer"=5
set "3","printer"=2
set "3","disk"=6
set "3","memory"=3
set "3","laptop"=7
set "3","USB"=1

set %=zzTermCorrelate("A,"B)
for i="":$order("B(i)"):"" do
  . write i,"!
  for j="":$order("B(i,j)"):"" do
    .. write ?10,j," ",^B(i,j),"!
output:

USB
  computer 1
  disk 1
  laptop 1
  memory 1
  printer 1

computer
  USB 1
  data 1
  disk 2
  language 1
  laptop 3
  memory 2
  monitor 1
  printer 2
  program 2

data
  computer 1
  disk 1
  laptop 1
  memory
  program 1

disk
  USB 1
  computer 2
  disk 1
  language 1

Figure 23 - $zTermCorrelate()

7.6.10.2 $zzDocCorrelate(gblref1,gblref2,mthd,thrshld)

A square Document-Document matrix gblref2 is calculated from the Document-Term matrix
gblref1 according to method mthd (Cosine, Sim1, Dice, Jaccard). The value of elements in the
Document-Document matrix will not exceed threshold (thrshld) and the cells associated with
coresponding document numbers will not exist.

A Document-Document matrix has document id's as its row and column indices. A cell in the
matrix indicates the degree to which the row document is related to the column document. The
diagonal is ignored. Figure 24 gives an example.

7.6.11 Stop and Synonym Functions

7.6.11.1 $zStopInit(arg)

7.6.11.2 $zStopLookup(word)

7.6.11.3 $zSynInit(fileName)

7.6.11.4 $zSynLookup(word)
A call to $\text{szStopInit}(\text{file\_name})$ will open and load a file of stop words into a C++ container. The file should consist of one word per line. If the file cannot be opened or there is insufficient memory to hold the list of words, the program will halt with an error message. $\text{szStopInit}()$ converts all words to lower case.

```mumps
#!/usr/bin/mumps
kill ^A,"B
set ^A("1","computer")=5
set ^A("1","data")=2
set ^A("1","program")=6
set ^A("1","disk")=3
set ^A("1","laptop")=7
set ^A("1","monitor")=1
set ^A("2","computer")=5
set ^A("2","printer")=2
set ^A("2","program")=6
set ^A("2","memory")=3
set ^A("2","laptop")=7
set ^A("2","language")=1
set ^A("3","computer")=5
set ^A("3","printer")=2
set ^A("3","disk")=6
set ^A("3","memory")=3
set ^A("3","laptop")=7
set ^A("3","USB")=1
set %=$\text{zzDocCorrelate}(\text{^A,^B","Cosine"},.5)
for i="":$order(^B(i)):"" do
.. write i,!
for j="":$order(^B(i,j)):"" do
.. write 10,j,"",^B(i,j),!
```

**Figure 24 - $\text{zzDocCorrelate}()$**

A call to $\text{szStopLookup}(\text{word})$ will return 1 if \text{word} is in the stop list, 0 otherwise. Words presented to $\text{szStopLookup}(\text{word})$ should be in lower case.

$\text{SzSynInit}()$ opens a synonym file. The file should consist of two or more words per line separated by from one another by one blank. The words are treated as synonyms with the first word on each line as the primary synonym. The primary synonym may be a code or category number. This word or code will be returned if any of the remaining words are passed as arguments to $\text{SzSynLookup}()$. Figure 25 gives an example.

7.7 SQL functions

These functions are peculiar to M2 and covered in greater detail in the distribution.

7.7.1 $\text{szsql}$

Returns the SQL server error message for the most recent command or 'ok.'

7.7.2 $\text{szsqlCols}$

Returns a string consisting of the columns names for the most recent operation that returned tuples. Each name is separated from the next by a TAB character ($\text{char}(9)$).
Assume that the file “stop” contains the word “and”

set %=$zStopInit("stop")
if $zStopLookup("and") write "yes",!

Writes yes

Assume that the file “synonyms” contains a line with the text:
compression compressions compress compressed compresses

set %=$zSynInit("synonyms")
write $zSynLookup("compressions"),!

output:
compression

Figure 25 - Stop list functions

7.7.3 $zsqlOpen

Returns true if a connection to the SQL server is open, false otherwise.
8 Pattern Matching

8.1 Mumps 95 Pattern Matching

Author: Matthew Lockner

Mumps 95 compliant pattern matching (the '?' operator) is implemented in this compiler/interpreter as given by the following grammar:

- Pattern Matching

```
pattern         ::= {pattern_atom}
pattern_atom    ::= count pattern_element
count           ::= int | '.' | int '.' | int '.' int
pattern_element ::= pattern_code {pattern_code} | string | alternation
pattern_code    ::= 'A' | 'C' | 'E' | 'L' | 'N' | 'P' | 'U'
alternation     ::= '(' pattern_atom {',' pattern_atom} ')
```

The largest difference between the current and previous standard is the introduction of the alternation construct, an extension that works as in other popular regular expressions implementations. It allows for one of many possible pattern fragments to match a given portion of subject text.

A string literal must be quoted. Also note that alternations are only allowed to contain pattern atoms and not full patterns; while this is a possible shortcoming, it is in accordance with the standard. It is a trivial matter to extend alternations to the ability to contain full patterns, and this may be implemented upon sufficient demand.

Pattern matching is supported by the Perl-Compatible Regular Expressions library (PCRE). Mumps patterns are translated via a recursive-descent parser in the Mumps library into a form consistent with Perl regular expressions, where PCRE then does the actual work of matching. Internally, much of this translation is simple character-level transliteration (substituting '|' for the comma in alternation lists, for example). Pattern code sequences are supported using the POSIX character classes supported in PCRE and are mostly intuitive, with the possible exception of 'E', which is substituted with 

```
[[:print]:[:cntrl:]]
```

Currently, this construct should cover the ASCII 7-bit character set (lower ASCII).

Due to the heavy string-handling requirements of the pattern translation process, this module uses a separate set of string-handling functions built on top of the C standard string functions, using no dynamic memory allocation and fixed-length buffers for all operations whose length is given by the constant STR_MAX in `sysparms.h`. If an operation overflows during the execution of a Mumps compiled binary, a diagnostic is output to `stderr` and the program terminates. If such termination occurs too frequently, simply increase the value of STR_MAX.

8.2 Using Perl Regular Expressions

Author: Matthew Lockner

In addition to Mumps 95 pattern matching using the '?' operator, it is also possible to perform pattern matching against Perl regular expressions via the `perlmatch` function. Support for this functionality is provided by the Perl-Compatible Regular Expressions library (PCRE), which supports a majority of the functionality found in Perl's regular expression engine.

The `perlmatch` function works in a somewhat similar fashion to the '?' operator. It is provided with a subject string and a Perl pattern against which to match the subject. The result of the function is boolean and may be used in boolean expression contexts such as the "If" statement.

Some subtleties that differ significantly from Mumps pattern matching should be noted:

1. A Mumps match expects that the pattern will match against the entire subject string, in that successful matching implies that no characters are left unmatched even if the pattern matched against an initial segment of the subject string. Using `perlmatch`, it is sufficient that the entire Perl pattern matches an initial segment of the subject string to return a successful match.

2. The `perlmatch` function has the side effect of creating variables in the local symbol table to hold backreferences, the equivalent concept of $1, $2, $3, ... in Perl. Up to nine backreferences are currently supported, and can be accessed through the same naming scheme as Perl ($1 through $9). These variables remain defined up to a subsequent call to...
\textit{perlmatch}, at which point they are replaced by the backreferences captured from that invocation. Undefined backreferences are cleared between invocations; that is, if a match operation captured five backreferences, then $6$ through $9$ will contain the null string.

Examples

This program asks the user to input a telephone number. If the data entered looks like a valid telephone number, it extracts and prints the area code portion using a backreference; otherwise, it prints a failure message and exits.

\begin{verbatim}
Zmain
Write "Please enter a telephone number:",!
Read phonenum
If $$^perlmatch(phonenum, "^(1-)?\(?\d{3}\)?\(?|-|\)?\d{3}-?\d{4}|$") Do
  Write "+++ This looks like a phone number.",!
  Write "The area code is: ",$2,!
Else  Do
  Write "--- This didn't look like a phone number.",!
Halt
\end{verbatim}

The output of several sample runs of the program follows:

Please enter a telephone number:
1-123-555-4567
+++ This looks like a phone number.
The area code is: 123

Please enter a telephone number:
(123)-555-1234
+++ This looks like a phone number.
The area code is: (123)

Please enter a telephone number:
(123) 555-0987
+++ This looks like a phone number.
The area code is: (123)

As in Perl, sections of the regular expression contained in parentheses define what is contained in the backreferences following a match operation. The backreference variables are named in a left-to-right order with respect to the expression, meaning that $1$ is assigned the portion matched against the leftmost parenthesized section of the regular expression, with further references assigned names in increasing order. For a much more in-depth treatment of the subject of Perl regular expressions, refer to the \textit{perlr} manpage distributed with the Perl language (also widely available online).
9 Mumps interpreter for Windows

From version 13.05, the code can be built (configure) and compiled (make/make install) in Cygwin.

The build scripts for Cygwin are:

1. BuildMumpsWithGlobalsInMySQLCygwin.script

2. BuildMumpsWithGlobalsInNativeCygwin.script

These build mumps.exe in a Windows based Cygwin environment that stores the globals in either the native Btree or a MySQL database. The mumps.exe file may be used in a normal Command Prompt window if the appropriate Cygwin DLLs are accessible. These are:

cygcrypto-1.0.0.dll
cygpcrc-1.dll
cygstdc++-6.dll
cygz.dll
cygmymysqlclient-18.dll
cygssl-1.0.0.dll
cygwin1.dll
cygcc_s-1.dll

These are available for free from:

http://cygwin.com/index.html

The Cygwin DLL files are covered by the GPL license as is this Mumps interpreter. See the Cygwin web site for details.
Alphabetical Index

absolute value.................................................................................................................. 50
Access via a Web Server................................................................................................... 41
ACID...................................................................................................................................... 9
Added builtin SQL Variables........................................................................................... 22
alignment.............................................................................................................................. 63
Apache.................................................................................................................................. 17
arc cosine............................................................................................................................. 50
arc sine................................................................................................................................. 50
arc tangent........................................................................................................................... 51
Array Index Collating Sequence....................................................................................... 45
ASCII................................................................................................................................. 52
backreferences.................................................................................................................... 44
Begin Transaction............................................................................................................. 9
blanks in numeric strings................................................................................................. 52, 53
Blanks in numeric strings............................................................................................... 44
Boyer-Moore-Gosper Function......................................................................................... 59
btree..................................................................................................................................... 53
buffer................................................................................................................................... 55
buffers................................................................................................................................. 53
buildClientServer.script................................................................................................. 7
BuildMumpsWithGlobalsInMySQL.script....................................................................... 7
BuildMumpsWithGlobalsInMySQLCygwin.script........................................................... 7, 70
BuildMumpsWithGlobalsInNative.script......................................................................... 7
BuildMumpsWithGlobalsInNativeCygwin.script............................................................. 7, 70
BuildMumpsWithGlobalsInPostgreSQL.script............................................................... 7
BuildNativeClientServer.script....................................................................................... 7
C++ container................................................................................................................... 66
cache hit ratio.................................................................................................................... 53
centroid vector.................................................................................................................. 57
cgi-bin.................................................................................................................................. 53
Client server common errors........................................................................................... 45
commit.................................................................................................................................. 23
Commit............................................................................................................................... 9
Compiling Large Programs............................................................................................ 47
Configure Options........................................................................................................... 5
Correlation Functions..................................................................................................... 5
Correlation Functions..................................................................................................... 64
cosine................................................................................................................................... 51
Cosine................................................................................................................................... 58
cygwin................................................................................................................................... 49
date...................................................................................................................................... 50
database............................................................................................................................ 70
date...................................................................................................................................... 49
date...................................................................................................................................... 50
date...................................................................................................................................... 51
date...................................................................................................................................... 52
date...................................................................................................................................... 53
date...................................................................................................................................... 53
Date functions................................................................................................................... 51
Debian................................................................................................................................... 5
Dice...................................................................................................................................... 58
document vectors............................................................................................................. 58
document.matrix.............................................................................................................. 65
dump................................................................................................................................... 52
dump file............................................................................................................................. 53
Examples.......................................................................................................................... 23
exponent.............................................................................................................................. 51
exponential......................................................................................................................... 51
file....................................................................................................................................... 53
File Names........................................................................................................................ 45
File names containing directory information............................................................... 45
file pointer.......................................................................................................................... 53
files..................................................................................................................................... 49
Format SQL Table............................................................................................................ 21
ftello................................................................................................................................. 54
General Relational Database Options.............................................................................. 5
Global Array Access....................................................................................................... 19
global array tree and data files....................................................................................... 49
Gregorian........................................................................................................................... 52
Gregorian date................................................................................................................... 51
HTML................................................................................................................................... 53
Implementation Notes..................................................................................................... 44
internal buffer................................................................................................................... 54, 55
Inverse Document Frequency score............................................................................... 64
January 1, 1970.............................................................................................................. 51
Job command................................................................................................................... 44
Julian date........................................................................................................................... 51, 52
left justify.......................................................................................................................... 53
Linux time.......................................................................................................................... 51
Lock command in client/server mode............................................................................ 44
Lock command with PostgreSQL................................................................................... 44
logarithm............................................................................................................................ 51
logarithms........................................................................................................................... 51
Math Functions............................................................................................................... 50
$zp\text{pow}$
$z\text{Replace}$
$z\text{restore}$
$z\text{restore[}$
$z\text{seek}$
$z\text{Shred}$
$z\text{ShredQuery}$
$z\text{sin}$
$z\text{SmithWaterman}$
$z\text{sql}$
$z\text{sqlCols}$
$z\text{sqlOpen}$
$z\text{sqrt}$
$z\text{system}$
$z\text{table}$
$z\text{tabsize}$
$z\text{tanh}$
$z\text{tell}$
$z\text{ui}$
$z\text{zw}$
$z\text{zwn}$
$z\text{zwp}$
$z\text{zws}$
$z\text{zws(string)}$
$z\text{zz}$
$z\text{zzAvg}$
$z\text{zzBMGSearch}$
$z\text{zzCosine}$
$z\text{zzCount}$
$z\text{zzDice}$
$z\text{zzDocCorrelate}$
$z\text{zzInput(var)}$
$z\text{zzJaccard}$
$z\text{zzMax}$
$z\text{zzMin}$
$z\text{zzMultiply}$
$z\text{zzScan}$
$z\text{zzScanAlnum}$
$z\text{zzSim1}$
$z\text{zzSoundex(s1)}$
$z\text{zzSum}$
$z\text{zzTermCorrelate}$
$z\text{zzTranspose}$